2020

Reliable Navigation for SUAS in Complex Indoor Environments

Andrew J. Fabian
Virginia Commonwealth University

VIRGINIA COMMONWEALTH UNIVERSITY

DOCTORAL THESIS

Reliable Navigation for SUAS in Complex Indoor Environments

Author:

Andy FABIAN

Advisors:

Dr. Robert KLENKE
Dr. Bartosz KRAWCZYK
Dr. Bridget THOMSON-MCINNES
Dr. Carl ELKS
Dr. Tim BAKKER

A thesis submitted in fulfillment of the requirements

for the degree of Doctor of Philosophy

in the

Collaborative Unmanned Aerial Vehicles Lab

Department of Electrical and Computer Engineering

December 3, 2020

VIRGINIA COMMONWEALTH UNIVERSITY

Abstract

Reliable Navigation for SUAS in Complex Indoor Environments

by Andy FABIAN

Indoor environments are a particular challenge for Unmanned Aerial Vehicles (UAVs). Effective navigation through these GPS-denied environments require alternative localization systems, as well as methods of sensing and avoiding obstacles while remaining on-task. Additionally, the relatively small clearances and human presence characteristic of indoor spaces necessitates a higher level of precision and adaptability than is common in traditional UAV flight planning and execution. This research blends the optimization of individual technologies, such as state estimation and environmental sensing, with system integration and high-level operational planning.

The combination of AprilTag visual markers, multi-camera Visual Odometry, and IMU data can be used to create a robust state estimator that describes position, velocity, and rotation of a multicopter within an indoor environment. However these data sources have unique, nonlinear characteristics that should be understood to effectively plan for their usage in an automated environment. The research described herein begins by analyzing the unique characteristics of these data streams in order to create a highly-accurate, fault-tolerant state estimator.

Upon this foundation, the system built, tested, and described herein uses Visual Markers as navigation anchors, visual odometry for motion estimation and control, and then uses depth sensors to maintain an up-to-date map of the UAV's immediate surroundings. It develops and continually refines navigable routes through a novel combination of pre-defined and sensory environmental data. Emphasis is put on the real-world development and testing of the system, through discussion of computational resource management and risk reduction.

Acknowledgements

This project would not have been possible without the generous support of the Electric Power Research Institute (EPRI), Virginia Commonwealth University (VCU) and Commonwealth Center for Advanced Manufacturing (CCAM). Their interest in developing technologies that cross boundaries between industrial machinery and unmanned flight, and their willingness to support high-risk research in this area, is inspirational.

Dr. Robert Klenke has my eternal gratitude for the opportunities he has given me. His leadership, mentorship, and selfless dedication to his students set a shining, aspirational example for any and all to follow.

I would like to thank the members of my committee: Dr. Tim Bakker, Dr. Carl Elks, Dr. Bridget Thomson-McInnes, and Dr. Bartosz Krawczyk for their insightful comments and thoughtful discussion. My peers at VCU: Peter Truslow, Matt Gelber, Kevin Yang, and Dr. Matt Leccadito have also been an invaluable source of advice and have helped me through numerous hours of flight testing and debugging.

Finally, I must express my profound gratitude to my family for providing me with unfailing support and continuous encouragement through my years of study. They're the best.

Contents

List of Figures

List of Tables

List of Abbreviations

FCS	**F**light **C**ontrol **S**ystem
GCS	**G**round **C**ontrol **S**tation
VNS	**V**isual **N**avigation **S**ystem
KF	**K**alman **F**ilter
EKF	**E**xtended **K**alman **F**ilter
RPY	**R**oll, **P**itch, **Y**aw (axis order and direction)
FRD	**F**ront, **R**ight, **D**own (axis order and direction)
NED	**N**orth, **E**ast, **D**own (axis order and direction)

For my parents. For showing what real strength is.

Chapter 1

Introduction

The history of UAV research and development has favored outdoor environments. Open spaces are test-flight friendly, and loud and unsafe machines such as drones need to be isolated from uninvolved people and workspaces. However many potential applications of drones are indoors: power plants can be inspected [94], factory operations can be automated [27], and security rounds can be made automatic, to name a few. Thus indoor, close-quarters navigation has seen a recent surge in research and commercial interest.

Indoor navigation introduces distinct new challenges. State estimation must achieve much higher precision, especially with regard to position and altitude estimation [24] if the drone is to use its position sensing to safely move within feet or inches of obstacles. It often cannot be assumed that the environment is static, since equipment and furniture can be moved frequently, and people and/or other machines may be moving through the space. Given this dynamic environment, it should be taken as a new requirement that the drone has sensory capabilities to rapidly observe its surroundings, and avoid collisions (at a minimum) while preferably rerouting around obstructions without affecting its mission plan.

There is an additional reason to be excited about these new demands: once met, a new level of autonomy is unleashed, where operators no longer need to plan around obstacles or double-check routes for clearance. The operator can essentially say "go there" to the drone, and let the drone figure out what doorways to go through and paths to take to get to the destination. As it moves, it constantly identifies unforeseen features of its environment and reacts appropriately, stopping or re-routing when necessary.

The research described herein was conceived and executed to offer a solution to a specific set of use-cases for drones. Conversely there are goals and benchmarks that make frequent appearances in indoor UAV navigation papers which are specifically *not* claimed here, as they are not aligned with the foundational use-cases. The foundational use-cases are repetitive observation and measurement tasks in industrial settings, which are currently performed by humans. For example:

- Visual inspection of the containment wall in a nuclear power plant. Per regulations this be done in sufficient detail to identify and locate sub-millimeter

cracks. Drones are a great fit for this task, since it is repetitive, requires image analysis, and occurs in a radiation-controlled zone where any human operations have additional costs and safety concerns. [82]

- Leak detection in high-pressure piping. Leaks can be detected via ultrasonic audio emissions when a suitable sensor is swept along the length of the pipe. A drone is an ideal candidate for this task, not only because of its repetitive nature, but also for its ability to move safely through cramped, hot, or otherwise unsafe spaces [95].

- Unusual vibration detection in factories. Prior to failure, motors often change their acoustic/vibration signature. By comparing these signatures over time, this can be detected. A drone could systematically canvas the factor floor to store and compare vibrations [44]. This mechanism allows immanent failures to be detected among all installed machines, without requiring physical modification.

Routine inspection of industrial equipment is notoriously error-prone, with human error as the main culprit [109]. Research on the use of computer vision to assist inspection tasks has existed for several centuries now [108], but is hamstrung by the need to acquire imagery. The ability to reliably and automatically operate flying cameras for data collection could be immediately paired with existing computer vision research to improve the quality of industrial inspections.

The use-cases stated above have several common denominators. First, they are well-known, planned and controlled indoor spaces. For this reason, Simultaneous Localization and Mapping (SLAM) systems are discounted from the solution as their principal strength - the ability to map out unknown environments on the fly - isn't needed, and is resource-intensive. Preparation of the environment is not seen as a hindrance, given the highly-engineered nature of these environments to begin with. Second, these environments are generally secure facilities, meaning that physical security of any equipment developed is assured by the encompassing physical security of the facility. Thus, access controls are not needed at the software level. If someone can physically reach the control station, they are presumed to be allowed to use it. Third and finally, transient obstructions will still exist, despite the regular nature of

these facilities. Whether people walking about, or equipment temporarily installed, any system navigating through these facilities must have an ability to perceive the world around it and safely move through it.

1.1 Contributions of the Dissertation

This research centers around the design, implementation, and testing of a UAV navigation system tailored towards indoor industrial applications. The Visual Navigation System (VNS), as it shall be known, was designed from the ground up with indoor-appropriate sensors, compute hardware, flight control software and navigation software, and custom GUIs for visualizing data and commanding the drone. It has been successfully tested in multiple locations. Although many individual aspects of the system could studied and improved, the system as a whole represents a feasible way forward for low-cost, highly agile multirotor drones operating in closed environments.

Four primary contributions are presented. Collectively, these contributions constitute a dependable system for indoor flight and navigation in close proximity to obstacles and humans.

1.1.1 AprilTag Localization System

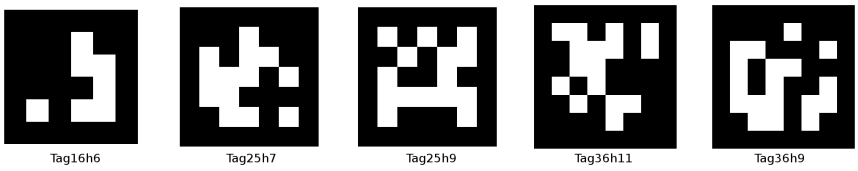

FIGURE 1.1: AprilTag Visual Markers

First, an optimized algorithm is described in Section 3.2 for obtaining position and pose from a set of visual markers, observed by one or more cameras. This system processes images in order to identify AprilTag markers in the frame; find the pixel coordinates of the tag's four corners; and then use those coordinates to estimate the camera's pose relative to the tag. The resulting transformation and rotation solutions are then placed into a global coordinate system to estimate the camera's position and

attitude. The end-goal of this component is to produce maximally-accurate position, pose, and uncertainty estimates, based solely on AprilTag markers.

Note that this is not the ultimate position estimate, which will come from a state estimation system and considers multiple data sources.

Adjacent to this section, a detailed study of the error characteristics of AprilTag-based position measurements is presented, concluding in a formula for estimating the variance of each sample based on relevant factors.

1.1.2 Full Visual Localization System

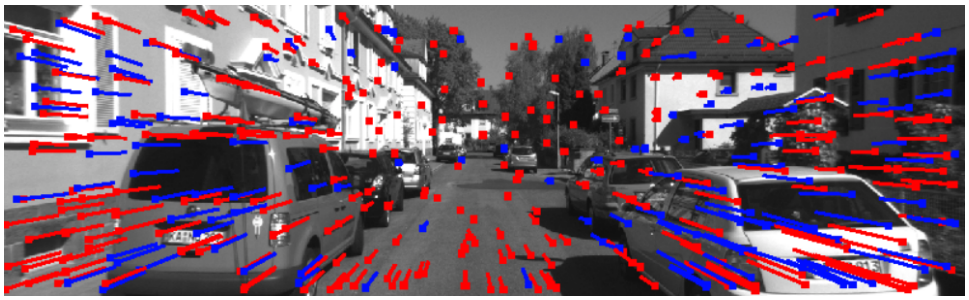

FIGURE 1.2: Visual Odometry

Second, and building upon the first contribution, a full localization system is designed and implemented. Recognizing that the AprilTag system is "blind" any time a tag is not in view, the full localizer is augmented by incorporating a 3D Visual Odometry (VO) system. By using VO for instantaneous velocity estimation, not only are all filtered position solutions improved, but the system continues to function in "blind spots" via dead reckoning, where no AprilTags are visible. Although dead reckoning is traditionally associated with unbounded drift, it may be possible to maintain accurate positions over time by using persistent visual features discovered in the environment.

Section 4.1 and 4.2 describe the testing done to quantify the performance of the localization system components, and the system as a whole, using a custom-made quadcopter test vehicle.

The litmus test for the success of this phase was defined to be that a drone would be able to takeoff, hold position, move to waypoints, and land in an obstacle-free indoor environment. This has been demonstrated, and is documented in section 4.9.

1.1.3 Collision Avoidance

Third, collision avoidance is demonstrated. The system must always refuse to move into detected obstacles, and additionally must be able to continue its mission by developing routes around obstacles on the fly.

Using the existing depth-mapping sensor, depth data can be accumulated to identify nearby obstacles. Other sensing solutions, such as millimeter-wave radar, are also suitable for this application, but were not chosen for this project.

Navigation is closely linked to this topic in two aspects. First, the drones have limited fields of view, and so algorithms were developed to control movement such that the drone uses its field-of-view as best it can to observe nearby visual markers, oncoming obstacles, or both. Second, the drone navigates under the assumption that small obstacles will present themselves regularly, and should be avoided without major disruptions to the planned mission. A navigation system based on mixing pre-planned, live, and pre-planned-but-updated information is discussed.

There are volumes of existing research on collision avoidance for UAVs, however the particular focus on indoor environments and 3D modeling excludes much of it. Sensory solutions are not considered here, and neither are transponder or ADS-B-type solutions. Research specific to SLAM advancements are similarly excluded, as that is a large field and out-of-scope for this project (although simply using a SLAM library is not). By dividing the collision-avoidance problem into two sub-problems - detection and routing, literature survey can be simplified to focus on obstacle detection using our chosen sensors. Routing, while necessary, is not seen as an area of novel research within the proposal body of work.

1.1.4 Long-term Data / Map Management

Finally, the system demonstrates long-term awareness of the environment. Plans are presented such that the system does not lose or have to "re-learn" information from flight to flight.

The system's "map" (its view of the world around it, which also informs the operator's UI display) incorporates both known data in a format comfortable to the operator (for example architectural drawings), along with discovered data. The user-supplied map contains absolute, non-volatile limits of movement, while obstacles discovered in-flight are always considered transient, to be re-examined whenever in view of the cameras.

This opens the door for further management interfaces. For example, an operator could define additional no-fly zones through the management computer, or define preferred paths that drones must stick to whenever possible. If combined with automatically recharging stations, an operator could run a fleet of drones indefinitely on changing tasks without physical interaction.

1.2 Organization of the Dissertation

The remainder of this dissertation is organized as follows: Chapter 2 presents a summary of prior work in the field of indoor UAV navigation, with a discussion of the benefits and drawbacks of each family of technologies. Chapter 3 methodically describes the entire Visual Navigation System designed under this research topic. All major components of the system are presented from a high-level design view, and then their implementation is described in finer detail. Extensive testing and analysis was performed to understand the performance of the sensors, subsystems, and ultimately the entire functioning Visual Navigation System. This testing and the results thereof are described in Chapter 4. Chapter 5 contains operator-focused notes to pass on practical lessons learned from this endeavour. Finally, chapter 6 concludes this work with a look forwards to possible extensions, improvements, and applications of the ideas presented.

Several appendices contain work that was judged to be adjacent to the topic at hand, but not a direct contributor to the main narrative. Appendix A describes testing done on the accuracy of the AprilTag system as a source of position and angle data in the face of varying lighting, tag size, receiver position, and camera settings. Appendix B covers expirements done to determine the stability of the test vehicle in flight. Appendix C presents an analysis of GPS position accuracy as measured by a

typical UBlox receiver. Finally, Appendix D discusses an early design decision that was revisited: switching from Python to C++ for the core system implementation.

Chapter 2

Background and Prior Work

2.1 State Estimation

State Estimation is the process of estimating the true state of a system from various observations of that system [75]. Observations have unavoidable shortcomings such as noise, quantization error, temperature dependence, and Nyquist frequency limitations. Therefore raw observations cannot be equated with a true state identification. With this challenge in mind, the field of state estimation focuses on using algorithms to estimate the true state of a system by combining multiple observations, often from multiple sources. State estimation also addresses the implicit problem of states that cannot be measured directly.

As a mature field [11], state estimation has several "best practice" solutions that are commonly-used. These are the Complimentary Filter, Wiener Filter, Kalman Filter, and Particle Filter. Current research generally seeks incremental improvements in the precision or robustness of one of more of those solutions, rather than proposing entirely new methods of state estimation. Additional current research proposes application-specific implementations of one of more of these estimators. This is a worthy topic, for although the estimators themselves are generic mathematical models, the limitations they impose on data sources and outputs often require extensive pre-processing of data in manners that are defined by the system at hand.

2.1.1 Complimentary Filters

Complimentary Filters form the basis of the most straight forward state estimation. It is often the case that complimentary pairs of sensors can be found to measure a signal: one with accurate low-frequency content, and one with accurate high-frequency content [45]. Position estimation is a good example: GPS receivers are accurate at low frequencies including DC, making them suitable as a basis for position estimation. However they do not react quickly enough to capture millisecond-scale movements. On the other hand, accelerometers can be sampled extremely quickly to output high-resolution position deltas, but suffer unbounded drift problems. A complimentary filter can use the low-frequency data from a GPS receiver and high-frequency data from an accelerometer to output position estimates with the accuracy

of GPS and responsiveness of accelerometers [102]. Complementary filters are simple, do not have failure modes, and are distortion-less [63].

Comparisons between complimentary filters and other state estimators exist in literature [76], though results vary widely. Complimentary filters are a reduced case the Kalman Filter and Wiener Filter [11], so one would expect their performance to be within the performance of similar Kalman and Wiener filters.

Marantos et al. notably have bucked the trend in current filter research by proposing an Attitude and Heading Reference System (AHRS) for highly dynamic vehicles based upon simple complimentary filters [64].

2.1.2 Wiener Filters

Weiner Filters [127] improve upon Complimentary Filters with a liberal drizzling of statistics. Whereas complimentary filters treat observations as simple numbers, Wiener filters assume them to be Gaussian-distributed probabilities. This retains the dependability of a complimentary filter, while accounting for the fact that observations can have known and varying amounts of error. Wiener filters are also a reduced case of the Kalman filter discussed below, where the Kalman Filter's covariance matrix is steady-state [45].

2.1.3 Kalman Filters

Kalman Filters (KFs) are the workhorses of modern state estimation. Kalman Filters extend the idea of Wiener Filter by adding a covariance matrix, which allows the filter to estimate not only the output states and their variances, but also cross-variances between all inputs and internal states [6]. For example, the system could determine that position sensors become less reliable at high speed or when turning sharply [31]. By requiring inputs and outputs to be linear and Gaussian-distributed, Kalman Filters can be implemented efficiently as matrix equations, although this restriction has also spurred the developed of more flexible alternatives: the Extended Kalman Filter (EKF) and the Unscented Kalman Filter (UKF). Finally, Kalman Filters introduce the possibility of system failures, known as divergence [99], when inappropriate feedback loops between the states and covarience matrix cause states to

"run away" without bound. Practical implementations of Kalman Filters often check for this failure mode and reset the filter if it occurs. However this should also be a trigger for the developer to look for missing information in their system model that would explain the state estimator's divergence from reality [116].

Kalman, Wiener, and Complimentary Filters are recursive filters. It should be noted that they never re-examine old data but always build upon previous iterations. The Particle Filter, by contrast, has a unique ability to reconsider past data after the incorporation of new information.

In 2017 Islam et al. created parallel complimentary and Kalman AHRS implementations for comparison [49]. They concluded, in agreement with prevailing wisdom, that KF had better noise rejection, but was more complicated to implement.

Other recent research has focused on improving the envelope of inputs for which stability is assured. [46] is a recently-published paper on this theme as applied to AHRS sensors, and particularly focusing on anti-windup for bias estimation as part of the system state. Similarly, [77] looks to apply physical limits to state transformations using the Rauch-Tung-Striebel method. In [50], Jenson describes a Multiplicative EKF which aspires to have a maximum input region with known stability.

A technique to estimate inputs as well as state variables using a Kalman Filter is described in [38]. Finally, Trimpe et al. [115] focuses on state estimation with non-constant communication from sensors, wherein missing observations are predicted to determine if state variations are sufficient to require new data. The authors describe their work as a new type of Ricatti equation in the field of controlled communication. Ricatti equations are standardized forms of first-order, nonlinear differential equations. A particular Ricatti equation, such as the one described in this paper, is defined by its functions $a(x)$, $b(x)$, and $c(x)$, which appear in the equation as coefficients to powers of x. This work can be used to save communication bandwidth on sensor buses.

2.1.4 Particle Filters

Particle Filters are conceptually distinct from the preceding chain of filters. They are based upon the idea of maintain a set of hypotheses for what state the system could be in at the moment, and choosing the best hypothesis to fit the current and

recent data. Note that unlike the preceding filters, the system can "switch" states between differing hypothesis in the presence of new data. Particle Filters tend to be computationally expensive, as a large number of particles is needed for acceptable results.

Given the recent popularity of UAVs, it's not surprising to see a Particle Filter-based implementation of an AHRS using MEMS IMU and GPS sensors [91]. This research also presents a competing Kalman Filter for comparison.

Researchers have also looked at combining the Kalman and Particle filters, to create a hybrid state estimator that is conceptually a Kalman filter, but with the Particle Filter's ability to escape local minima. [52]. Alternatively, Qi Cheng et al. propose a Particle Filter containing internal banks of Kalman Filters to improve individual particle positions in [20].

Finally, Particle filters are notoriously resource-intensive due to the large number of particles required for acceptable results. Compounding the problem, solutions typically err on the side of caution by over-specifying the number of particles. Wang et. al attempt to both determine and optimize the number of particles required in such a filter. Their approach tracks a target using a Kalman filter, reducing the search area for the particle filter. This has the additional benefit of allowing the system to predict the targets position during periods of time when sensors are occluded [125].

2.1.5 Application-Specific State Estimation

Numerous works are being published on the topic of state estimation as applied to a particular application, often of commercial significance. Recent publications are summarized in Table 2.1, and are further described below.

Author	Application	Sensors	Algorithm(s)
Ilyas	Micro Planetary Rovers	IMUs	EKF, UKF
Gao	Generic Strapdown Navigation System	IMU, GNSS	CKF
de Marina	UAV Attitude Estimation	IMU	UKF, FOAM
Ko	UAV Navigation	IMU, GNSS, Barometer	IEKF
Feng	Indoor Positioning and Navigation	IMU, UWB	EKF, UKF
Guo	Small Solar-Powered UAV	IMU, GNSS	EKF
Du	UAV in Multiple Environments	IMU, GNSS, Optical Flow, LiDAR, RGB-D	EKF
Yang	Small UAV	IMU, GNSS	EKF

TABLE 2.1: Application-specific research on state estimator implementations

Ilyas et al. describe the state estimation used aboard a theoretical micro planetary rover [48]. The authors point out that relative motion sensing can be performed using familiar technologies from earth-bound navigation, however absolute positioning requires novel sensors due to the lack of a GPS constellation. The availability or lack of a planetary magnetic field as a source of heading and/or attitude information is not discussed. As a replacement, a sun sensor is proposed. A comparison is made between EKF and UKF implementations merging data from the sun sensor, gyroscopes and accelerometers. Raw attitude measurements from the sun sensor are also plotted. The authors conclude that the EKF and UKF implementations have almost identical performance, both of which are substantially better than raw attitude information. Given the extremely slow motion of planetary rovers, and corresponding lack of dynamic forces on the vehicle, one would expect any state estimation algorithm to yield good results.

Gao et al. address the issue of asynchronous data between multiple sensors in [36]. In particular, two-way communications between Beidou transceivers and satellites incur variable latency on the order of hundreds of milliseconds. A Kalman Filter variant titled the Cubature Kalman Filter is used and described in this work, but the

timing issue is addressed by including the Beidou latency as a state to be estimated and updated. Results suggest that this system improves position accuracy over more basic systems, however the results bear additional scrutiny as the reported position errors are an order of magnitude over what is to be expected from a Beidou receiver.

de Marina [65], Ko [53], Guo [41], and Yang [129] all present state estimators for outdoor UAVs. de Marina's and Guo's systems are more basic, describing novel implementations of standard sensors and Kalman filters. Ko pushes state estimation up to the individual sensors, attaching state estimators based upon an "Invarient" EKF which is described in their paper. Ko compares the results of the IEKF state estimator to that of the open-source ECL EKL estimator, and find that performance is similar. Of note, the ECL EKF is used by VCU in the Aries flight controller. Finally, Yang

Moving to the arena of indoor navigation, Feng [33] examines one of the candidate systems for the principal body of work in this paper: Ultra-wideband RF transceivers plus IMU data. UWB transceivers, as examined in 2.4.3 can be used as a source of 2D or 3D position data. Interestingly, Feng proposes a positioning solution using only a single UWB base station combined with an IMU, rather than the typical three base stations and least-squares position estimation. This novel system is tested against the typical system.

Finally an ambitious Hao Du published research on sensor fusion from a multitude of sensors for a vehicle capable of traversing multiple indoor and outdoor environments [28]. This recent article documents an EKF designed around GPS, IMU, Optical Flow, LiDAR, and RGB-D cameras. As a side effect of this complex setup, the prototype multirotor described in this article weighs nearly 20 lbs, and carries a full Intel PC onboard! Flight tests shows that the extra sensors made the system resiliant against GPS data glitches.

One can conclude from these applications that some form of non-linear Kalman filter (EKF or UKF) is the ubiquitous choice for small-vehicle state estimation. Although the product of this research described in 3.4 uses a Wiener-based state estimator, it can be concluded based upon this survey and upon the identify of Wiener filters as special-case Kalman filters that a non-linear Kalman Filter would be an ideal choice.

2.2 Visual Markers

Visual markers and visual navigation are an ongoing area of research interest and publication. This section surveys the state of research for several topics within this arena: Visual Markers, Marker-based Localization Systems, Marker-based Landing Systems, and ajacent computer vision research.

2.2.1 Visual Marker Designs

Many designs have been created for two-dimensional visual fiducial markers. Designs vary by purpose: "2D Barcodes" are designed to be information-dense when scanned at close range under controlled conditions. Other markers are designed to facilitate position and orientation detection and only carry minimal information - typically a small ID number.

ARToolkit [5] is one of the earliest systems for fiducial markers which was broadly adopted by academic research. As the name implies, this Augmented Reality Toolkit contains many components for AR research, including visual markers. ARToolkit markers can be distinguished by their graphically-designed interior spaces, which must satisfy certain conditions related to rotational symmetry, but are otherwise arbitrary. ARTag [35] also focused on bad-light conditions and low false-positives by switching from arbitrary inner graphics to digitally-coded grids. This allowed it to also add checksumming and forward error correction (FEC). Figure 2.1 shows ARToolkit and ARTag markers.

FIGURE 2.1: ARToolkit & ARTag Markers

The InterSense Circular Tag [73] (Figure 2.2) was a commercial product developed in parallel with ARToolkit. Circular tags have advantages for position detection, but lack enough correlatory points for pose estimation, although this can be remedied with multiple tags. Naimark's paper focuses on edge detection and refinement for subpixel-accurate center coordinates. InterSense has developed a mature product line around the Circular

FIGURE 2.2: InterSense Marker

Tag including sensors and network interfaces, and markets them for military flight simulators.

The SCR [135] marker is designed Siemens, the industrial machinery company. SCR support has been interated into Siemens visual sensor products and programmable logic controllers (PLCs). Details are scarce on the specific benefits of this design, and it seems to be limited to coarse localization of a user, where the user either can see a marker, or cannot.

FIGURE 2.3: Siemens SCR Marker

Aruco markers (Figure 2.4) were idealized implementation of square markers: scalable size based to dictionary length, maximum inter-word distance. [37]. In 2020 improvements were proposed [123] to increase distance precision by adding extra circles around the marker. The Aruco system contains a family of marker vocabularies, rather than a single set of markers. A specific marker set can be chosen based on the dictionary size needed and basic shape of the marker. In addition to maximizing the inter-word distance, which combats mis-labeled detections, the Aruco system prescribes a method for generating markers while maximizing the number of bit transitions in each marker, to facilitate edge detection.

FIGURE 2.4: Aruco Marker

AprilTag [80] and AprilTag 2 [122] (Figure 2.5) were created by the University of Michigan to optimize detection in uneven lighting. They have been broadly, though not exclusively, adopted for indoor UAV and robotics applications. Version 2 uses the same tags, but with rewritten software to improve performance on small images, lower computational time, and emit fewer false-positives. Version 1 had a strong focus on successfully detecting partially occluded markers, but this feature was found to be lacking demand, while the accompanying false-positive detections were a significant problem.

FIGURE 2.5: AprilTag Marker

The excessively confident author of [83] poses the question, "what is the best fiducial?" and then makes up a new fiducial to answer that question. This marker (Figure 2.6), which has not been widely adopted, is unique in that it uses grayscale

gradients in its core. Design decisions such as shape, color, and detection false-positive and false-negative conditions are discussed.

Finally, in the category of information-dense markers, Quick Response (QR) Codes [39] are the current de-facto standard for high-bandwidth tags, often displayed to the public for consumption by smartphone applications. The data payload within one marker can be URLs or text up to 4K characters, although the marker size and complexity will increase with payload size. QR Codes need significant processing power both to detect and decode.

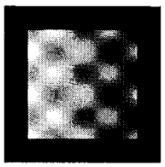

FIGURE 2.6: Owen's Marker

MaxiCode [1] was created and is used by the United Parcel Service (UPS). Each tag stores 93 characters of info. This system has been standardized as ISO/IEC 16023, and includes Reed-Solomon error correction.

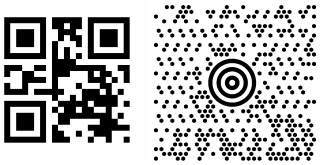

FIGURE 2.7: QR and MaxiCode Markers

2.2.2 Localization Systems

Beyond the raw ingredients of computer vision and some system of visual markers or features, position and motion estimation is a complex topic with many possible solutions. The most straightforward adaptations of commonplace technology for this purpose use visual markers to create position samples, and input those samples into a Kalman Filter for integration with other sensory data, as demonstrated in the works [133] and [136]. Other systems use SLAM-like technologies to discover the position of new markers: [7] and [128]. An open-source position system, "OpenKai" is tested in [61] using a ZED stereo camera for visual odometry data, which is send to ArduPilot's state estimator. The test noted that this system works better with nearby objects, and that same yaw-to-translation coupling was observed.

A novel concept is explored in [3], where fiducials are mounted on rovers which communicate with airborne drones. While in-between tasks, the rovers move to reset the position references to locations appropriate for the next task. The rovers

get their positions from the drones when moving, and the drones get their positions from the rovers otherwise.

Finally, [60] must be mentioned for the impressive results obtained, even though it's not strictly a marker-based system. This paper develops dynamically feasible trajectories with large accelerations and rotations, and demonstrates this by asking a quadcopter to fly though a vertical slot. The task can only be accomplished if the quadcopter "throws" itself through the slot at a 90-degree roll angle. Visual odometry is used as part of the quadcopter's state estimation.

2.2.3 Landing Systems

Rather than informing position estimation, fiducial markers can be used to identify specific locations, which can be either stationary or mobile. A significant body of research focuses on the task of automating precisely-positioned landings, by using visual markers to identify the landing zone. When performing an automated landing, one encounters the problem of the camera inevitably closing in on the landing zone (LZ), until the LZ exceeds the camera's field-of-view. This may cause problems for visual marker detectors. Landing on a moving target presents additional complexity, as the target's state must be estimated and predicted.

In [40], the authors split landing into two phases: searching and approaching. The fiducial marker is placed vertically above the landing station, and the vehicle homes in on the target, and then descends. Marker detection is integrated into the ORB-SLAM [72] system. A different study, [51], focuses on the final stage of landing when the target exceeds the camera's field-of-view. The authors propose a new marker design based on concentric circles which can be detected and positioned, even when only partially visible. Similarly, in [124] a system for tracking a fixed landing zone using multiple big and small visual markers is described, and custom markers are created for this task, though the markers aren't significantly different than existing alternatives. Testing is done using a DJI Matrice 100.

Several reports document research implementations of marker-based landing systems. The first, [74], combines AprilTag and generic feature detection used for split position and LZ tracking with a moving LZ. The system is tested by attempting to land a Yamaha RMax on a boat, although the test did not complete the landing.

Next, [96] implements a system for fixed LZ tracking using visual marker, combined with an EKF for predictive target tracking. The system uses front and down-facing cameras, and is tested on an AR drone. A final, similar body of work in [89] implements mobile LZ tracking via visual marker and EKF, but is only tested in 3D simulations.

A single paper addresses the topic of fixed-wing UAV landing using visual markers. A system described in [66] places vertically-mounted fiducial markers on either side of the runway to aid the aircraft in maintaining its centerline and glideslope as it lands.

2.2.4 Related Vision-based UAV Research

Beyond fiducial markers, computer vision-based research adjacent to navigation is actively exploring multiple topics and approaches to solutions.

Formation flight is a popular topic, and is typically approached with a leader/-follower paradigm, where the leader is uniquely equipped with indicators designed to allow followers to determine the leader's relative position and attitude. Four papers published in the last nine years demonstrate varying techniques for this. In [62], the authors implement a system for fixed-wing formation flight. They prepare the leader airplane with LED beacons, and augment the followers with cameras and software to detect the LEDs. This demo simplifies the formation flight problem by using the localization system only to adjust the followers state estimate; there is no closed-loop control between leader and followers. Next, [110] uses a similar LED beacon and camera system. Their implementation uses a particle filter to deduce the leader's pose from collected samples. Their implementation has been tested using modified DJI quadcopters. Walter et al. describe a hybrid relative/absolute positioning system [120] based on UV markers - some of which are ground-based, and some of which are affixed to aircraft. This system was tested on hexacopters. In a related paper [121], the same author describes an improvement to the system by flashing the UV indicators to communicate ID numbers. Testing included a variety of lighting conditions, from dark indoor spaces to direct sunlight.

Although the bulk of visual navigation research is centered around either SLAM, fiducial markers, or visual beacons, a variety of more esoteric topics have generated

recent publications as well. Teuliere et al. create a 3D model [111] of the working environment, and then compare live images to that model in order to determine position and orientation. Comparison is done through edges, and so much of the paper is dedicated to appropriate techniques for segmentation and edge detection. Cardenaz has a similar approach [16] of following terrain by identifying edges in imagery and associating them with roads, rivers, or coasts. Since these edges aren't straight lines, the edge detection problem focuses on RANSAC spline fitting. Next, [118] attacks the localization problem by comparing captured images to pre-existing maps, using Normalized Cross-Correlation. Current testing indicates that their system works extremely well on indoor environments, and works correctly outdoors albeit with more error. In [21], the authors take a similarly human-inspired approach to localization by using landmarks that can be detected by through neural network classifiers. Finally, [71] proposes a system using lasers projected from the drone onto the ground, and [84] attacks the special case of corridor navigation, using vanishing point detection and a simple corridor model.

2.2.5 Comparison & Adjacent Papers

The following research publications are of interest to this subject, but are of disparate categories. Therefore they are presented in a list:

- [67] is a comparison paper wherein three methods of vision-based localization are deployed on quadcopters and compared for accuracy and notable characteristics. The methods are: IR LEDs, Colored Balls, and an "H" Landing Pad. The paper determines that IR Leds are fast, but require external hardware. Tennis balls are fast, but depend on color accuracy. The "H" target is slower, but more accurate on the normal axis. In landing tests, the "H" system got closest to the target coords. All three had performance acceptable for autopilot.

- [87] is a detailed analysis of marker error versus distance and viewing angle. This data contains the same local maxima in angular error at nearly straight-on viewing angles as was observed elsewhere in this research.

- [70] proposes a framework for generating maps of predicted visual navigation performance across an area. Besides visualization, Merzic discusses the use of

this data in path planning applications. As presented, the data doesn't incorporate the effects of vehicle heading, which could be significant.

- [134] is a strict comparison targeting four metrics: usability, efficiency, accuracy, reliability. This comparison is done based upon ideal markers only, and so is focused on the limits of the various algorithms, rather than real-world performance limitations.

2.3 Path Finding

The following discussion summarizes the state of research with regards to pathfinding algorithms, with the caveat that its scope is limited to topics that are fundamental or directly related to the Navigation System presented in this paper. The system implemented in this paper is a hierarchical A* system, with a non-homogeneous hierarchy. Two popular topics are omitted due to this scope. First, multi-agent pathfinding is out-of-scope. Second, optimizations that assume the special case where the navigation graph is rectangular grid are similarly ignored, although this is a popular line of inquiry.

This discussion follows the evolution of accepted pathfinding algorithms through three cornerstones: Dijkstra's 1959 algorithm, the A* ("A Star") algorithm, and finally hierarchical implementations of A*.

2.3.1 Algorithms

Dijkstra's Algorithm

In 1959, Edsger Dijkstra published his seminal work [25] describing and solving the problem of finding a minimum-length path between any two nodes in a graph. Dijkstra's algorithm is essentially a breadth-first search that sums path lengths as it moves. Figure 2.8 describes the algorithm in detail.

1. Assign the starting node a distance of zero. Call this the current node.
2. For each neighbor of the current node, assign that neighbor a distance, which is defined to be the distance from the current node to the neighbor plus the current node's recorded distance. If the neighbor already has a distance, keep the smaller of the two.
3. Move to the node with the lowest distance, and repeat the process.
4. Close/remove nodes when all neighbors have been explored without reaching the goal.
5. Once the goal is reached, the path used to reach the goal is the correct result.

FIGURE 2.8: Dijkstra's Pathfinding Algorithm

When visualized, Dijkstra's algorithm can be seen to search outward from the starting point in all directions, such that there is an expanding wave-front that eventually crosses the destination point. Implementations of this algorithm have been studied and optimized extensively. Choice of data structures, such as priority queues rather than arrays, can be shown to decrease the average-case and worst-case execution time. Neither time is particularly good; this algorithms strength is in its simplicity and intuitive operation. It's time complexity is often stated as $O(n^2)$ [106].

A*

Building upon Dijkstra's algorithm, the A* [43] algorithm capitalizes on the fact that the general direction of the target can be known even if the exact path to it has not been discovered. A* requires a heuristic function to estimate the remaining cost to reach the target from any point, and moves in the direction with the best estimated remaining cost. This function can be defined in any manner the author chooses, as long as it meets certain criteria. Often, the euclidean distance to the target point is used as the heuristic.

This heuristic system allows A* to prefer to move towards the target, rather than in all directions. This technique can backfire, particularly in maze-like maps where the best path often moves away from the target. However the average case is still better, giving A* an $O(nlogn)$ complexity [98].

1. Assign the starting node a distance of zero. Call this the current node.
2. For each neighbor of the current node, assign that neighbor a distance, which is defined to be the distance from the current node to the neighbor plus the current node's recorded distance. If the neighbor already has a distance, keep the smaller of the two.
3. **For each neighbor of the current node, compute the sum of the neighbor's accumulated distance and the heuristic remaining-distance function.**
4. Move to the node with the lowest **accumulated+remaining** distance, and repeat the process.
5. Close/remove nodes when all neighbors have been explored without reaching the goal.
6. Once the goal is reached, the path used to reach the goal is the correct result.

FIGURE 2.9: A* Pathfinding Algorithm

One should note that Dijkstra's algorithm is a special case of A*, where the heuristic function is a constant.

A* is a de-facto standard pathfinding algorithm, and has spawned a large body of research dedicated to incrementally improving A* in various ways. Comprehensive and up-to-date surveys of A* research exist [119] [2] [8], but some of the more established A* variants will be discussed below to provide some familiarity with this field.

"Iterative Deepening A* (IDA*)" [55] addresses the trade off between depth-first

and breadth-first searching. Depth-first searching can have unpredictable timing, and isn't guaranteed to find the best result. Breadth-first search, which includes both Dijkstra and A*, is memory-intensive but admissible. The proposed IDA* algorithm applies a known search technique, Depth-First Iterative Deepening, to the A* algorithm. As the name suggests, this method uses repeated depth-first searches with limited but increasing maximum depth. Although this results in redundant work when the lower depths are searched repeatedly, performance is found to be similar to A* in test cases. Memory usage is improved, since A* requires space for open- and closed-set lists potentially equal to the size of the graph. Meanwhile IDA* only requires memory to store the current best path, and normal stack space.

"Dynamic A* (D*)" [104] approaches the problem of a changing environment. This algorithm stores path solutions, and also stores changes to the environment / graph, such that it is able to calculate updated solutions at lower cost than the initial solution. D* is significantly more complex than A*, and imposes restrictions on how the environment is allowed to change. Specifically, the cost of graph edges are allowed to change, but the shape of the graph (nodes and edges) cannot. "D*Lite" [54] accomplishes the sames goals but has a unique and faster implementation.

"Learning, Real-time A* (LRTA*)" [14]. This algorithm combines two concepts: real-time search and dynamic heuristic determination. Real-time search algorithms are given a deadline to deliver *some* result by, even if it isn't the optimal result [56]. Meanwhile, dynamic heuristic determination algorithms learn and improve the heuristic function over time to optimize it for a specific graph. LRTA* is proposed as a best-of-breed algorithm built open related research on Learning, Real-time A* algorithms. It has particular application in video game development, where AI characters may need to determine routes in real-time, regardless of available processing power.

"Triangulation A*" [23] focuses on preparing the search graph from the environment in a manner than minimizes the number of nodes and edges. Rather than the naive approach of using every corner of every object as a node, this technique divides all of the open space into triangles, and then connects those triangles with nodes. This graph can then be simplified by eliminating by replacing linear series of nodes with only the beginning and ending nodes. As the performance of all

pathfinding algorithms is dependant on graph size, this optimization can be applied to any of the above A* variants.

Finally, approaches exist that pre-compute all possible routes through a map, and then store them to a database [9]. Real-time pathfinding becomes a simple key lookup, which is vanishingly fast. The downside to the approach is that the database, which is typically kept in memory, can become prohibitively large. Thus data compression techniques are investigated to decrease the memory footprint for path databases.

Hierarchical Approaches

The methods discussed so far have in common the trait that they all compute complete solutions in one pass. As a consequence they all require complete navigation graphs, and incur the storage and computation costs associated with larger graphs. As these graphs continue to grow, this family of solutions inevitably hits scalability limits. Responding to this, a family of hierarchical algorithms was developed, wherein successive levels of details are used to to find an approximate solution, and then refine it within the local areas it touches. There are many variants of this technique, including the one presented in this paper in Section 3.3.

"Partial Refinement A*" [107], "Heirarchical Pathfinding A*" [10] [57], "Hierarchical Path Finding" [29] [86], and "HPA*" [10] are examples of this type of algorithm. The authors of [107] offer the metaphor of humans planning a cross-country trip, noting that the initial plan can be developed quickly, by choosing major highways that make up the bulk of the trip. Details about how to get to and from those highways are determined as a second pass, and may be able to be delayed until much later in the trip. Additionally, changes to the trip, such as pulling off a highway for a rest stop, only requires updates to that local segment of navigation. The big-picture plan still survives intact.

There is overlap in multiple papers between the ideas of hierarchical levels-of-detail for graphs, and the idea of precomputing solutions to paths. In general, the idea of precomputed solutions for all paths is assumed to be prohibitively expensive as an $O(n^2)$ problem. However, with simplified maps, either through reduced detail,

or through reduced area, n may shrink to a suitable small number. This concept is explored in detail in several separate implementations: [97], [130], and [42].

Dynamic path planning methods, meaning methods that can replan against an updated graph more efficiently than from-scratch, are of interest to this research, as the ultimate project includes navigating an environment which is constantly being sensed and refined. "Anytime Dynamic A*" [59] takes this approach, while also implementing the real-time pathfinding technique of initial and successively more accurate outputs.

Finally, a pair of papers [19] [18] focus on the data structure under which hierarchical graph data is stored. Quadtree-based approaches are stated to be faster, at the cost of yielding imperfect solutions. Through adjustments to the quadtree data structure, the authors are able to obtain fully-accurate solutions while maintaining the speed of quadtrees.

2.3.2 Aviation Applications

This final subsection focuses on application-specific implementations of pathfinding systems within the Air and Space fields.

A 2020 Master's Thesis paper describes an application built to develop ideal multi-hop routes for military aircraft, given their range, pilot limitations, and a database of suitable airports [34]. It is implemented as an Excel application with embedded VBScript, and uses the A* algorithm.

In [93], Rippel presents a highly customized path generation design for compatibility with general aviation (GA) flight. The system includes a kinematic flight model and GA-based constraints on altitude, terrain clearance, and runway patterns. This specialization allows the system to implement such concepts as climbing early to clear an eventual mountaintop. The paper delves into different pathfinding solutions, and ultimately uses a hierarchical Dijkstra approach.

Airbus contributed to [4], a paper on using optical control and pathfinding techniques to develop optimal descent paths, such that as an airplane approaches an airport to land, it's engines are at idle for the majority of the descent. This is done with an awareness of the airspace regulations typically in-play during an airliners descent to an airport. The paper compares results from this technique against results

obtained from a current Airbus simulator, and shows a 30% reduction in fuel usage during descent.

NASA's Mars Rover presents very unique navigation problems in [17]. The current navigation algorithm is "susceptible to failure when clusters of closely spaced, nontraversable rocks for extended obstacles." This paper results from a 2005 technology task developed to address this problem. The improved system described in this work is based on Field D*, a hierarchical, dynamic pathfinding algorithm. Potential problems are also described, such as sandy slopes causing the wheels to slip and the state estimate to be corrupted. Compounding this task, computation resources are limited, as this and 96 other applications must run aboard a radiation-hardened computer with a 20 MHz clock and 128M of RAM. The resulting system is physically tested in a duplicate Mars rover in a simulated environment at NASA's Jet Propulsion Laboratory.

2.4 Sensory Techniques for Indoor Localization

Though techniques for localization are numerous, they can be categorized into only a few basic algorithms. The actual computations needed to triangulate position from distance or angular measurements using the methods described below are well know and readily implemented, so a detailed discussion of them is omitted here. Localization is ultimately based on observations of distance or angle from reference points. These points may be fixed in space, fixed to a plane, or unknown. The distance to a point or the angle from a defined vector originating at the point can be measured. A set of these observations, varying in target point, source point, time, or any combination thereof can be used as a basis for a full localization solution. Observations can be classified as follows: [85]

Time of Arrival (TOA) - The receiver and base stations have synchronized clocks. One unit emits a signal at a known time, and the received time is measured.

Time Difference of Arrival (TDOA) - The receiver "pings" a base station, which responds quickly. The round-trip time is measured, and the distance is computed, compensating for any expected system latencies. This avoids the need for clock synchronization.

Received Signal Strength (RSS) - The signal-to-noise ratio of beacons is measured and assumed to represent the distance between units.

Angle of Arrival (AOA) - The system includes methods for measuring the angle between the receiver and base stations. Multiple angular constraints are discovered and combined to find a position.

Using these fundamental measurement techniques, several techniques have been explored as implementations of indoor localization: Fingerprinting, Distance Measurement, Visual Odometry, SLAM, and Visual Markers. Each of these will now be discussed in detail.

2.4.1 Fingerprinting

"Fingerprinting" refers to the collection of techniques used to recognize one's position by comparing sensory data to against previously-stored samples or environment models[132]. Sensory data is often based on existing radio communications:

WiFi networks and signal strengths, bluetooth devices, and/or raw RF traffic. Magnetic signatures, which are affected by ferrous building materials and electrical power systems, are also commonly used [103]. Best-in-class systems report accuracy in the meter-range.

Fingerprinting systems are subject to the limited accuracy of RSSI measurements (which in turn are affected by multipath interference and shadowing), and thus require de-noising filters, such as weighted centroid calculations or Kalman filters. Fingerprinting systems also incur the initial cost of the generation of a fingerprint database or environment model. Recent research has begun to tackle the problem of distinguishing mobile or temporary RF sources from permanent, fixed sources[137].

These systems are well-suited to operation on a cell phone, thus determining the location of the owner within a structure. For example, Carnegie Mellon University's "Handy Andy" system is designed towards identifying which room a user is in [101]. Use in an autonomous vehicle localization system is a trickier proposition, since 1-meter accuracy may not be sufficient for navigating through a cluttered environment. Nonetheless, exactly that is being proposed by Brzozowski et al. in a pair of papers focused on UAV navigation via magnetic field fingerprinting[12][13]. Brozozowski's research shows extensive visualization of the magnetic field in test areas, while also explaining the level of calibration needed to achieve useful results with a magnetometer. It concludes with the idea that magnetic fingerprinting may be an "important hint" for localization systems. (A 2011 paper does similar experiments with an outdoor, fixed-wing aircraft, and concludes that Wi-Fi RSSI fingerprinting is suitable for use in that application [103].)

2.4.2 On-board Distance Sensing Systems

Now we consider the class of systems where an on-board transceiver communicates with a set of stationary transceivers. Systems in this class use some method to determine distance from this communication, typically TDOA. The roving transceiver may "ping" each possible fixed station dozens of times per second, computing a full localization solution after each loop through the set of fixed stations.

2.4.3 Ultra-wideband Radio

With the introduction of DecaWave's DW1000 [22] single-chip Ultra-wideband transceiver in 2013, and subsequently the Pozyx localization kit [90] in 2016, UWB-based localization has become dramatically simplified.

In contrast to FM or AM radio, which vary the frequency or amplitude of a continuous sine wave RF emission, UWB radio sends short pulses over wide (bandwidth is > 20% of the center frequency) sections of the RF spectrum. It can be run at very low power levels, allowing it to co-exist with conventional radio traffic on the same frequencies. A principal benefit for positioning is that the system is not sensitive to multipath interference, since the RF pulses are short enough that the line-of-sight signals and reflected signals can be seen separately [117].

Much research and publication was done on the subject of building a quadcopter UAV that uses UWB for positioning, possibly augmented by a secondary source such as SLAM or GPS. The seven papers summarized in 2.2 and 2.3 all describe UWB-based localization system attached to indoor sUAS and flight-tested:

Paper	**UWB System**	**Other Systems**
Tiemann [113]	DWM1000	None
Li [58]	DW1000	Laser Scanner
Perez-Grau [88]	Unknown	SLAM
Tiemann [114]	Unknown	None
Shi [100]	DWM1000	None
Tiemann [112]	Unknown	SLAM
Hyun [47]	Unknown	Multipath RayTracing

TABLE 2.2: UWB Localization on sSUAS Literature Comparison - Hardware

Paper	2D Accuracy (cm)	3D Accuracy	Anchors Used	Flight Area (m^2)	2D or 3D
Tiemann	10	20	8	64	3D
Li	*unknown*	4	*unknown*	42	3D
Perez-Grau	16	19	3	40	3D
Tiemann	15	30	4	9	3D
Shi	50	50	8	15	3D
Tiemann	*unknown*	20	6	9	3D
Hyun	24	24	16	384	2D

TABLE 2.3: UWB Localization on sSUAS Literature Comparison - Accuracy and Test Environment

Experiments were done to determine the suitability of a Pozyx UWB localization system for use in indoor navigation. These experiments centered around three topics: accuracy, noise level, and disturbance analysis.

These system, at it's best, has fantastic accuracy. In a room-sized environment with four Pozyx markers installed, the system had an absolute accuracy of 105 mm across a 2D plane. 3D accuracy was not tested, as it would require more Pozyx markers than were available. It also has a fairly low noise level, measuring a standard deviation of 26 mm in the same area.

However problematic behaviors were found. The system's position measurement would drift slowly, changing position as much as 150 mm over several minutes. Additionally, when humans moved about the room, and particularly when they moved between the Pozyx beacons and the measurement unit, persistent offsets would be created in the data. Fig. 2.10 shows data captured from one such experiment. The unusual data on the second half of this measurement is the result of a person walking near the sensor. Both the beacons and sensor were kept stationary during the experiment, such that the expected result is a constant-value reading.

With respect to the deployment of a large-scale indoor navigation system, UWB, and particularly Decawave DW1000-based technologies have limited scalability, as the range of individual beacons is limited, each beacon requires a power source, and beacons time-share the same radio spectrum. UWB systems could however become attractive due to their inclusion of a unique communication channel. The

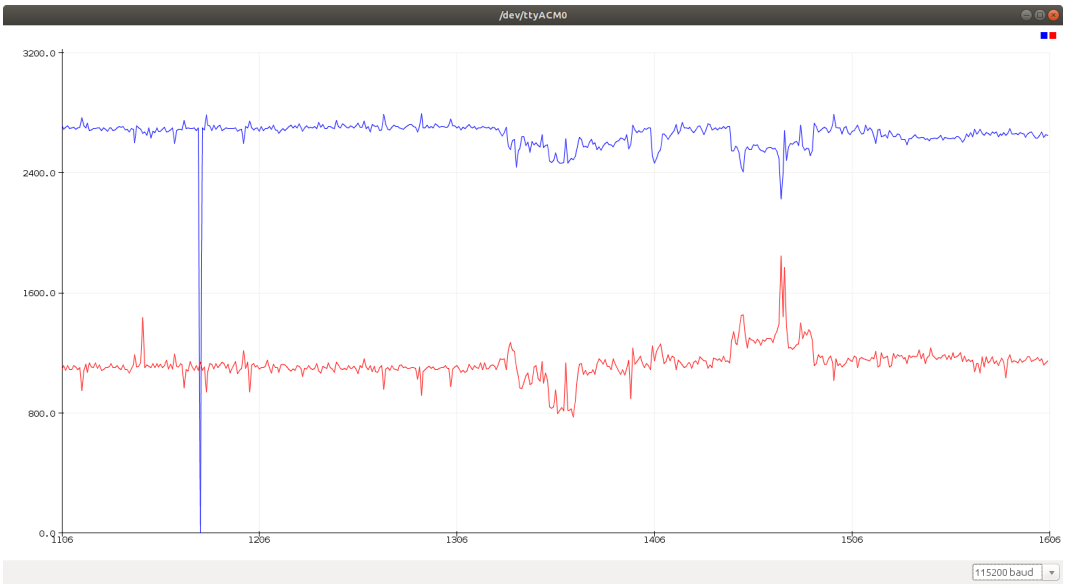

FIGURE 2.10: Pozyx testing data showing offsets caused by nearby human movement

DW1000 system, in addition to its localization functionality, has the ability to transfer arbitrary data at power levels below typical background RF levels.

2.4.4 External Motion Capture

Systems such as Vicon and Optitrack use a set of high-speed cameras to track objects in a scene. An external computer processes the images and outputs coordinates of target objects. For flight control, these coordinates can be wirelessly transmitted to the vehicle. These systems are known to be highly-accurate [69], and suitable for usage as a reference when testing other systems. They are expensive, and require extensive preparation.

These systems will be included in the full literature survey.

2.4.5 SLAM

This collection of systems uses cameras or other environmental sensors to build a map of their surroundings, while simultaneously using that map to track their position and attitude. The environment is typically processed into "features", or identifiable visual patterns, and the 3D position of those features is estimated. As time progresses, an expanding database of features is built. SLAM often includes the topic of

"Loop Closure", where drift accumulated by movement into new environments can be corrected once a known environment is re-seen. [15]

SLAM systems for sUAS indoor navigation will be included in the full literature survey.

2.4.6 Visual Markers

Visual Markers, such as Quick-Response (QR) Codes, are designed to be readily-identifiable by computer vision systems. These vision systems can often determine the orientation and position of the marker, relative to the camera. This can be used to locate the camera, and/or to locate mobile objects with affixed visual markers. For example, Boston Dynamics robots are seen lifting boxes which have large AprilTag codes printed on them. When used for positioning, this system relies on the camera having an unobstructed view of the marker, at a distance small enough that the individual "pixels" of the marker can be resolved. The image must also be crisp and relatively free of blur.

An early practical example of visual markers in flight is a 2011 paper [62], which demonstrates the use of high-intensity LED markers. These markers are detected by an on-board camera and processing system on a neighboring aircraft, and used to synchronize their two flight paths.

The AprilTag fiducial marker system was published in 2011 as well [80]. Design of computer-readable fiducial markers is an active area of investigation. QR Codes, for example, contain high data density, while AprilTags are designed for 6 degree-of-freedom pose estimation with a small data payload. AprilTag 2, the current version, was proposed in 2016 [122], and focuses on improving processing speed and recognizing targets in small images.

With regards to fiducial markers used for localization, 2017 saw the introduction of several papers using visual markers to identify particular points of interest. In [89], a system was developed to autonomously land on a visual marker, with robustness against temporary vision interruptions. Wang et al furthers this concept in [124] to include multiple markers of varying size in a hierarchical arrangement. This design allows a large marker to be used, so that it can be seen from a greater distance,

while switching to progressively smaller markers as the drone approaches the landing site. This is proposed as a solution to the issue where the visual marker grows beyond the drone's field-of-view and becomes unrecognizable when the drone approaches the visual marker. The first GPS-like, full-position sensor models were published in 2018 [136].

Chapter 3

Visual Navigation System Description

3.1 Components and Connections

Quadcopters are perhaps the perfect example of a 21st-century aircraft, due to their absence of any traditional mechanism for actually flying. Lacking wings, a tail section, neutral buoyancy, or helicopter blades (rotating wings), they can neither generate lift nor stabilize themselves via any aerodynamic means.

Instead, they are a completely reliant on embedded computer design, wherein their microprocessors use PID loops to generate artificial stability, filters to separate signals from noise, and motion planning to help the quadcopter move in a way that makes sense to humans. Even the simplest modern drone has no fewer than six onboard computers: a main flight controller, a digital radio transceiver, and a motor controller for each motor.

This chapter will present the design of the Visual Navigation System. This is presented both at the conceptual level of interconnecting software components, and at the practical level, which includes the specific hardware and configuration used in the test vehicle.

3.1.1 System Overview

The system will meet the following requirements, which are derived from the use-cases described in Chapter 1:

- Precise absolute position, heading, and attitude determination in a prepared environment
- Disallowed sensors: GPS, Magnetometer, Barometer
- Aircraft must be stabilized against drifting, regardless of the presence or absence of position data
- Aircraft must be able to detect and route around physical obstacles. It should "remember" the presence of these obstacles for future flight planning.
- Must work without exceeding resource limits on a large (warehouse-sized) area
- The aircraft must be capable of "point and click" autopilot control.

3.1.2 On-Board Systems

While the concepts of the Visual Navigation System are largely hardware-agnostic, it is useful to have a physical testbed to prove and demonstrate the concepts. This document describes the Visual Navigation System (VNS) system and the test drones created to aid in research and development.

The VNS described herein is composed of both typical and novel drone components and communication channels. The first fundamental layer - the air frame and propulsion system - includes the motors, speed controllers, and propellers. This covers most of the physical structure of the aircraft and components that transfer energy from the battery to the propellers. Two airframes were used in the development of this system, the DJI "Flamewheel" F450 frame [26], was used for initial development and later, the larger Tarot 650 airframe was used to allow for additional camera sensors and application payload sensors.

The low-level control of the aircraft is performed by the VCU's "Aries" [126] [30] flight controller computer. Aries handles both low- and high-level piloting functionality: it does "inner loop" stabilization, which uses high-speed gyroscope-based control loops to stabilize the aircraft. At the same time, Aries does high-level motion planning, motion control, and communications. As VCU-developed intellectual property, the Aries source code is available for modification to suit research projects, and was customized here to handle specific requirements of this work, for example, the ability to process a continuous stream of waypoint commands without "jerky" behavior.

Aries, when used in its typical outdoor environment, uses a GPS receiver for position information, a barometer for altitude data (since altitude correlates with a known transfer function to air pressure), and a 3-axis magnetometer to register compass headings. All of these were determined to be unsuitable for indoor operation: GPS signals would be weak or missing; air pressure is affected by HVAC systems, doors opening and closing, and ground effect; and a magnetometer will experience significant local interference due to ferrous building materials and AC power distribution.

Instead, new sensors and sensor fusion algorithms were needed for position,

altitude, heading, and velocity information. This need established the base requirements for the Visual Navigation System as described herein. The Visual Navigation System is the set of sensors, computational resources, and algorithms required to estimate position, velocity, nearby obstacles. It should also include a pathfinding component that can navigate through obstacles to reach a destination.

A ZED Stereo camera [131] can natively output instantaneous velocities and rotation rates in three dimensions, while also functioning as an imagery source. That imagery can be consumed by a Visual Marker detector and Point-n-Pose estimator, to provide position, altitude, and heading information. All of this requires substantial computing resources, especially the ZED Camera, which requires a CUDA-enabled GPU in addition to general-purpose Linux system. The NVidia Jetson TX2 [78] small form-factor computer was chosen for this application, as it represented the best ratio of performance to weight available at the time, and met all of the other system requirements.

The Jetson computer interfaces with the Aries flight controller via a serial port (Universal Asynchronous Receiver and Transmitter, or UART) configured to 115200 baud, giving an effective data rate of approximately 11.5 kilobytes per second in each direction. VCU Aerial Communications Standard (VACS) [68] packets are used to packetize and encode data flowing over this link. The full set of on-board communication paths is illustrated in Fig. 3.1 and shown installed in Fig. 3.2.

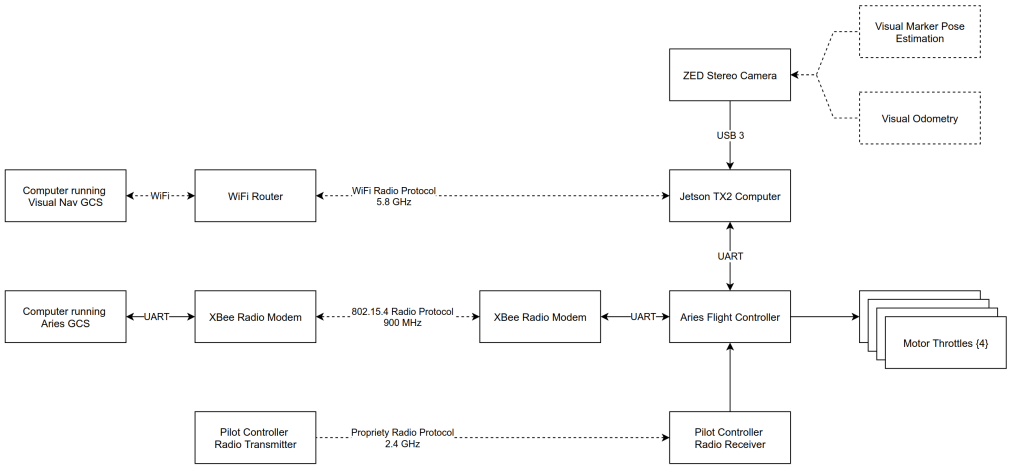

FIGURE 3.1: Data paths aboard aircraft and for air-to-ground communications

3.1.3 Mechanical Concerns

As an aircraft-bourne system, the Visual Navigation System must satisfy physical constants, in addition to it's functional requirements. Its weight must be sufficiently low that the aircraft can carry it while maintaining acceptable power-to-weight ratios and energy reserves to execute its mission. It must have a package size and mounting mechanism that is compatible with the aircraft. It's weight affects the aircraft's center-of-gravity (CG), which may adversely affect the aircraft's control system if the CG is not in the expected location. It must be tolerant of vibration and turbulent airflow, as both of these are produced by the aircraft during flight.

Conversely, the aircraft may affect the Visual Navigation System, either through vibration, or by interfering with sensors' field-of-view. Image sensors are particularly sensitive to vibration, which manifests itself as blurred or streaked images. The first mitigation to this should be steps taken to reduce vibration at the source, since that increases flight performance and efficiency in general. After that, physical isolation through rubber or foam mounts can be effective, as can photographic tools like maximizing shutter speed.

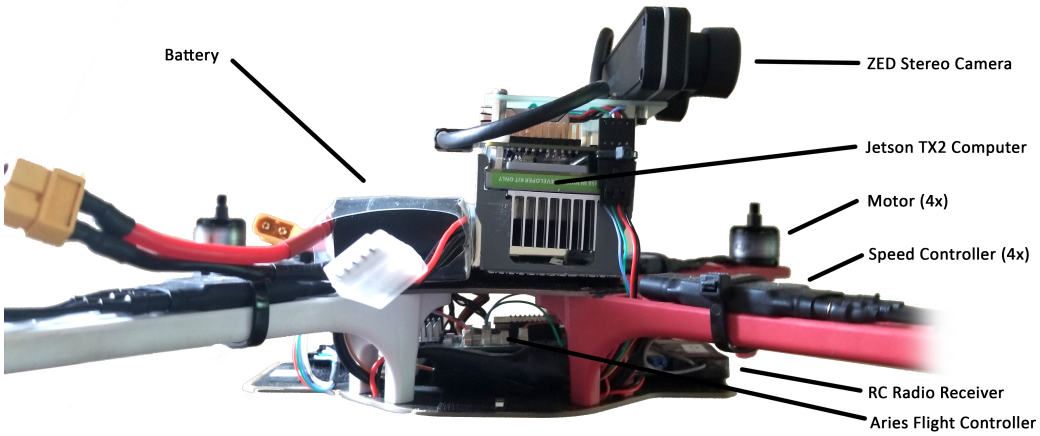

FIGURE 3.2: Test drone with flight control and indoor navigation components

Efficiency and Size are often at odds, and viewed as an engineering trade-off. Indoor navigation often demands a small physical footprint for the drone, so that it can navigate small spaces and fit through doors. On the other hand, large propellers generate substantially more lift from the same amount of energy, so longer flight times favor larger drones.

3.1.4 Ground Systems

Although the aircraft is capable of flying completely autonomously, with zero RF emissions or communications, a set of ground equipment and software has been in standard usage to configure, monitor, and debug the various on-board systems. The Aries flight controller has a companion application, called the Ground Control Station Lite (GCS-Lite), which communicates with the Aries flight control over a wireless UART link. GCS-Lite uses a map-based display to visualize the aircraft's position and heading, set waypoints, and monitor the aircraft's progress in flight. The display has superimposed instruments: a ground- and airspeed indicator, a vertical speed indicator, an altitude indicator, and an artificial horizon. To the left of the map, vital information is displayed through a combination of numeric readouts and color-coded indicators. Graphing functionality is also available, should it be more convenient to view these data as time-varying traces. Finally, the right third of the screen is home a set of configuration panels, listing all configurable parameters and commands available on the aircraft. A screenshot of the image is below in Fig. 3.3.

Development of the Visual Navigation System required re-tuning of several aspects of the Aries Flight Control System in response to the new sensory data available to it. Sensor delays and accuracy parameters are key to the successful operation of the Kalman filter-based state estimator. Though experimental tuning and testing it was found that the estimator behaves best when configured to strongly trust the velocity data emitted from the Visual Navigation System, weakly trust the position data, and account for the significant time delay in the Visual Navigation Data, which was found to be approximately 300ms. The altitude and position control loops were re-tuned by adjusting low-pass filter cutoff frequencies and PID controller coefficients. Having a GCS with a radiotelemetry link to the aircraft meant that this could be done in mid-flight, without having to land and re-program the drone for each iterative step. Extensive use was also made of the GCS's ability to graph internal data, such as signals before-and-after lowpass filtering, and the individual contributions of the Proportional, Integral, and Derivative components of a running PID controller.

A second piece of software was created for this project: a GUI specifically tailored

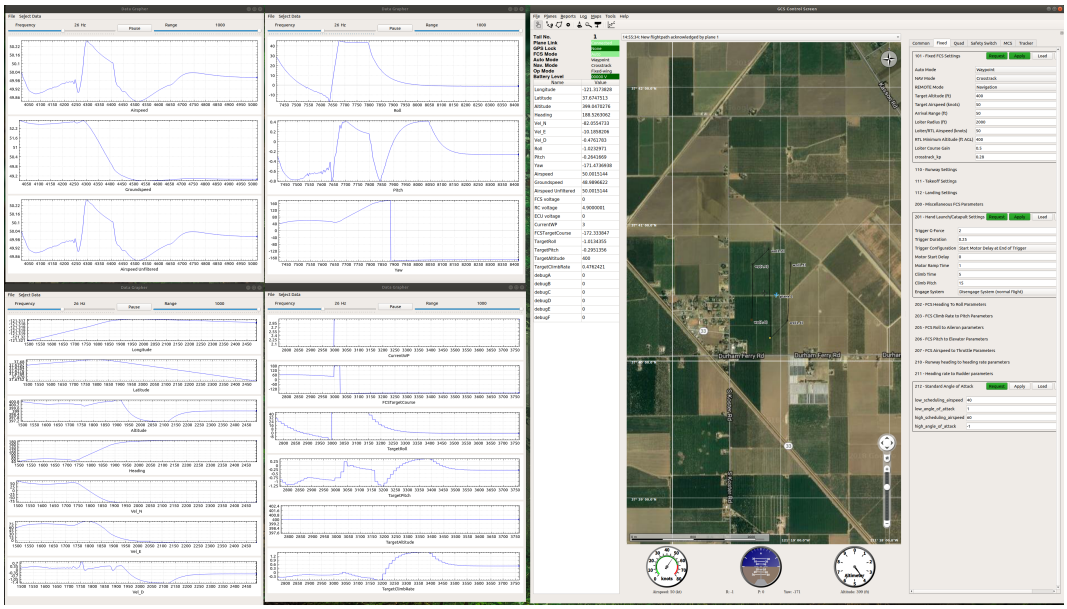

FIGURE 3.3: Screenshots of the Aries Ground Control Station (GCS)

to the visualization and debugging needs of the Visual Navigation System. Requirements were identified, including the need to visualize much smaller spaces in higher resolution, such that all geographic data could be displayed to sub-millimeter precision, and the map zooming and navigation controls could accommodate this depth of detail. The visualizer should be able to display the location of AprilTags in the environment, and visualize AprilTag detections. Crucially, it should be able to visualize the voxel field described herein, which is a three-dimensional occupancy map used for navigation. This necessitated a full 3D view with appropriate camera controls. Finally it should be able to visualize the three-dimensional undirected graphs generated as part of the pathfinding system. Due to the bandwidth needs of this collection of data, it is transmitted over a WiFi channel using a combination of UDF for telemetry, and HTTP over TCP for data that requires end-to-end integrity, such as the voxel field and navigation graphs. A data server application was developed in C++ to handle communications and short-term data storage, and a GUI was developed in Javascript, taking advantage of libraries such as THREE.js, and the cross-platform abilities inherent in web-based applications. The GUI is pictured below in Fig. 3.4.

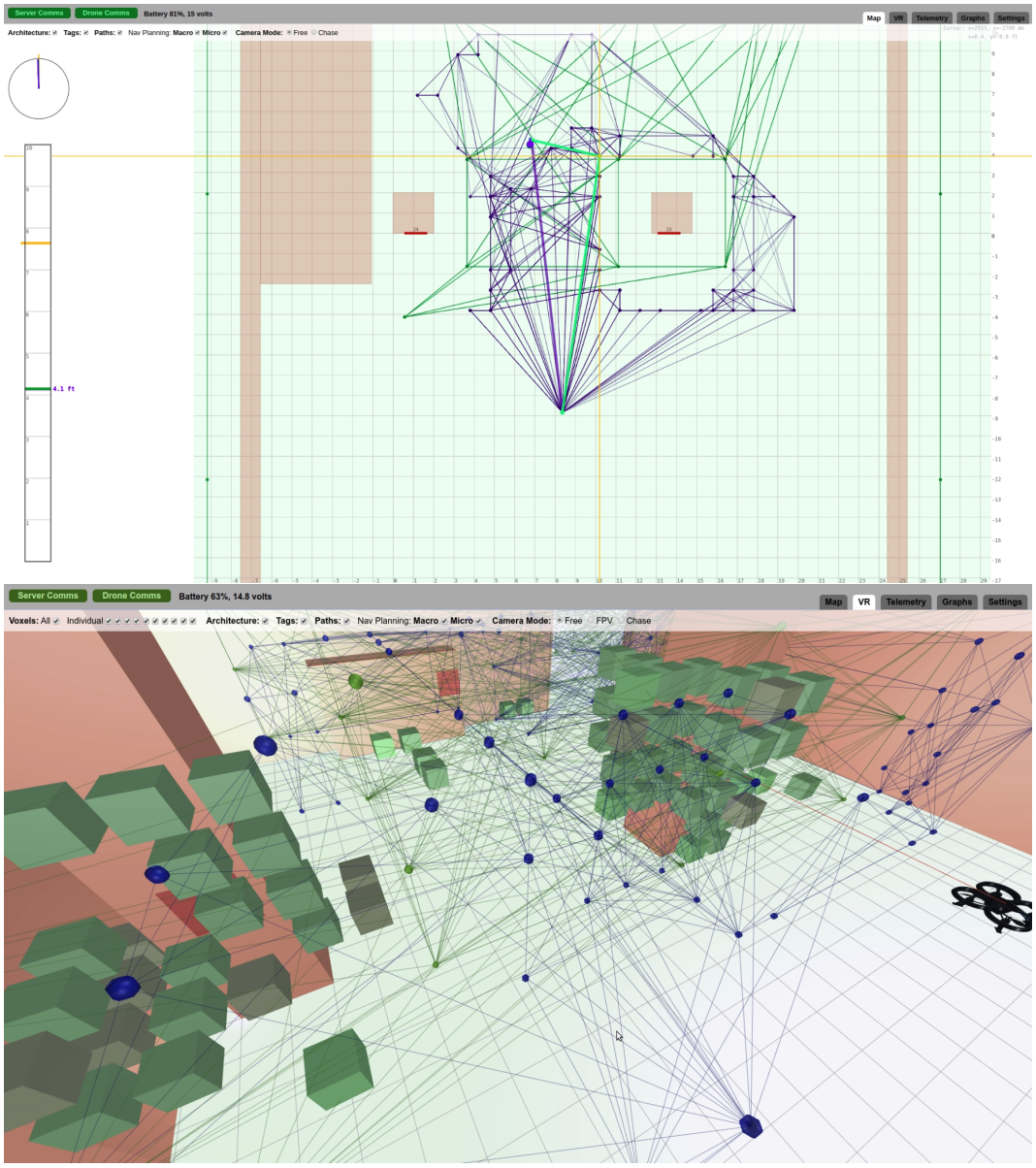

FIGURE 3.4: Screenshots of the Visual Navigation System GUI

3.2 Visual Marker Navigation

Optical fiducial markers are the backbone of the localization system used in this project. They have several advantages: they're inexpensive to produce; they can produce extremely high-resolution results; they can be mounted anywhere; they are passive devices which can be detected with common imagery systems. Although a myriad of fiducial marker technologies are now available, they generally follow the same design paradigm.

Chapter 5 describes the research that was done to compare fiducial markers against other forms of localization technology, and then to analyze the performance and limits of fiducial markers as a localization aid. The current chapter builds upon that foundation to describe the implementation of a working drone-carried localization system.

3.2.1 Image Generation and Pre-Processing

The pipeline starts with an image. As light passes through a cameras' lenses and apertures, it is cast upon an image sensor, which records the intensity of light striking each pixel of the sensor. Sensors are grayscale only, but by covering sensors with red, green, and blue filters, color images can be obtained.

By the time the pixels of an image are recorded by an image sensor, several decisions have already been made: focus has been set by positioning the cameras' lenses so that light coming from a particular range of distances is in-focus when it strikes the sensor. Light from all other distances is blurred to varying degrees, in what is known as the Bokeh effect. This applies equally to fixed-focus cameras; they are just pre-focused to cover the distance range deemed most useful. Decisions about light level are also now burned into the image, as those decisions control which pixel values are clipped to minimum/maximum values, resulting in permanent information loss. More specifically, the camera's shutter is opened for a certain amount of time to let light energy accumulate on the sensor, and the camera's aperture, if variable, is opened or closed varying degrees, also allowing more or less light into the camera. These decisions affect the brightness of every pixel in the image, and are important to choose correctly. During the analog-to-digital conversion of each pixel, where

brightness values are mapped to integers, floor and ceiling values are imposed such that any light levels below the floor are recorded as the same value (zero), and thus any additional differentiating information in those pixels is lost. The same limit and loss of information occurs at the maximum integer value. Neither clipped light levels, nor out-of-focus imagery, can generally be fixed through software processing.

Lenses also cause image distortion. The effect can be subtle in high-end photo and video cameras, or extremely pronounced in miniaturized one-piece cameras. Lens distortion can consist of radial and planar components. Radial distortion is typically the most prominent, and in extreme cases causes the "fish eye" effect, but generally can be identified by looking in the image for lines that ought be straight, but are actually curved. Linear distortion can be caused by a misalignment, either in position or angle, of the lens versus the image sensor.

Good imagery starts with selecting appropriate camera parameters to get the correct light levels into the sensor. As summarized in [32]: when visual markers are visible in the image, use a feedback loop to take their "black" and "white" pixels and drive them into the middle of the camera's brightness range. When no markers are visible, scan the entire frame and adjust light levels to create as few saturated pixels as possible. Fig. 3.5, taken from that research, shows that the histogram for a whole image can have markedly different means and extents when compared to a histogram of only the AprilTag ports of that image. Thus an AprilTag-aware image brightness control can be an improvement over a generic one.

Resolution is another consideration. While this is often as simple as "more is better", since AprilTag pose estimation in general benefits heavily from resolution, a system-aware approach to this decision might dictate different conclusions. For this author's test system, resolution is a positive factor for visual marker accuracy, but a negative factor for visual odometry frame rates and reliability, so a compromise resolution of 720p (1280 pixels wide by 720 pixels tall) is chosen. A full discussion of the tradeoffs between resolution, working distance, field-of-view, and marker size follows in Section 3.2.3.

Images must be de-warped in order to removed unwanted distortions introduced by the camera's lenses. Established systems are available for using test images to identify image warping. That information can be used to build a de-warping

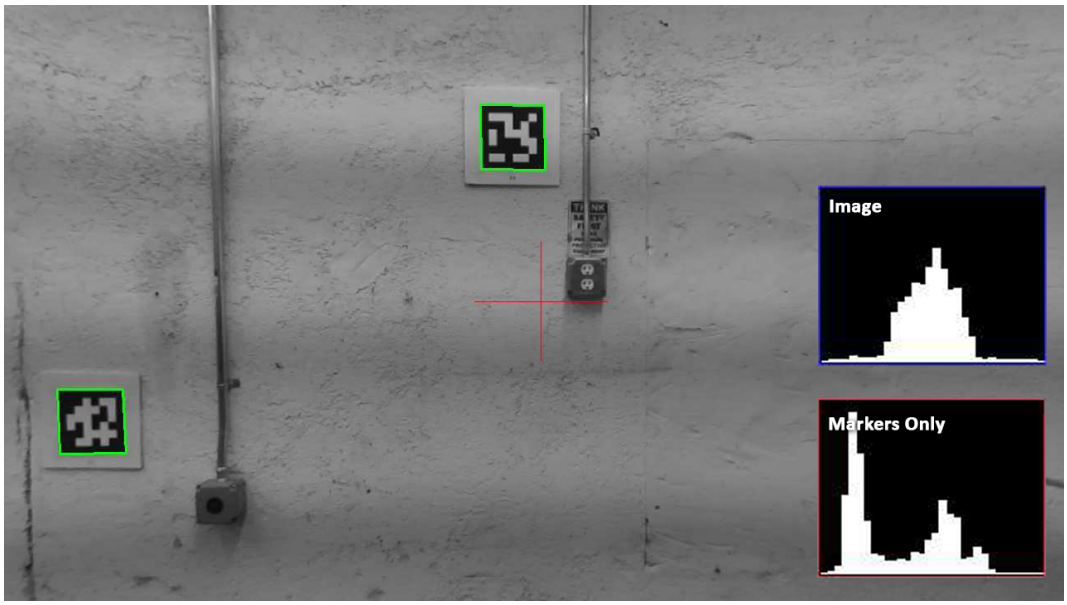

FIGURE 3.5: Histograms of Pixels with AprilTag contents vs. whole-image histogram)

function that generates an ideal (rectilinear) image from the original. Rectilinearity is critical when using imagery for position and angle measurements.

For this navigation application, the image can be converted to grayscale, since the AprilTag algorithms used in this project do not use color information. Other applications such as subject detection may want to retain color information.

3.2.2 Marker Detection and Pose Identification

Marker detection and pose identification is handled primarily by the AprilTag software library created by University of Michigan researchers Edwin Olson and John Wang [122] [81]. "Pose" in this context refers to the translation and rotation of the marker, relative to a coordinate system origin. Their two-paper series on AprilTags is the definitive reference on the design and implementation of an AprilTag detector.

The AprilTag system, including its software library, provides many options for tuning the system to a specific purpose and available hardware. Several "families" of tags are available, trading off address space size (number of IDs available) for complexity. This project uses the "tag36h11" family, which as an address space of 587 unique IDs.

The "quad decimate" setting helps to strike a balance between tag detection strength and resource usage. As one of the set of standard AprilTag settings, it controls an

optional downsampling step that can be applied to the input image. "1" means to keep every pixel, while "3" means to keep every third pixels on both axis, for example. Since images are two-dimensional, the amount of data to be processed reduces by approximately the square of this number, so a setting of "3" scales the workload down by a factor of 9. On the other hand, with only every third pixel available, the size of the smallest detectable code increases by a factor of three on each axis, or (stated differently) the distance at which a marker can be detected is reduced by a factor of 3. For this project, it was observed that there is a strong correlation between marker distance and loss of detection accuracy. Based on that result, obtaining maximum detection distance is not a priority for this project, as the data would be all but unusable anyway, and indeed would be mostly filtered out at the state estimation phase. Choosing a decimation factor of three still allows for detections at all useful distance/size combinations, which reducing processor loading to a level that permits the apriltag detector to run at a full 30 frames/second.

It should be noted that regardless of the value of this setting, the pose calculation is done at full image resolution for maximum accuracy.

The "nThreads" parameter is also significant for resource management, as it controls the number of CPU threads spawned by the AprilTag detector. These are short-lived threads that are created and destroyed for each image, so they create "bursty" loads that affect multiple CPU cores simultaneously. For small numbers of threads (1-4), performance increases significantly with each extra thread, though the effect tapers off towards larger thread counts.

The full set of AprilTag software settings as used by this system are listed below as Fig. 3.6.

Setting	Value
Tag Family	tag36h11
Quad Decimate	3
Quad Sigma	0
nThreads	3
debug	3
Refine Edges	1
Decode Sharpening	0

FIGURE 3.6: AprilTag library settings as programmed into the visual navigation test system

3.2.3 Error Computation and Thresholding

The AprilTag Point-and-Pose algorithm often outputs two possible solutions, rather than a single estimate. Due to the limited information available for its internal optimization problem, there are often two solutions that are equally likely, that visually appear as reflected about the center of the marker. Taking the average of the two is not a good option, since it's guaranteed to be wrong. If there exists a good quality position estimate at the moment (as maintained in the state estimator component described in Section 3.4), then the option that is closer to the current position estimate can be chosen. Otherwise both answers are discarded, and the system must wait for an iteration where there is only a single response.

Along with the position estimate, the state estimator requires the expected standard deviation of each sample. This is used to decide how heavily to "trust" this sample, compared to previous samples and compared to other data sources. Sample variance can be computed using information about the AprilTag's physical size, distance from the camera, angle from the camera, and the pose estimate's internal quality score.

As a visual method of communication, AprilTags are subject to visibility concerns: line-of-sight and illumination. As a medium captured through image sensors, AprilTags are additionally subject to photography concerns: pixel resolution, camera field-of-view, blur, and light level control. This section will address each of these concerns and quantify their effect on the performance of the standard[1] AprilTag library's Point-and-Pose measurement functions.

Minimum Resolution

It is evident that each pixel (black or white square) of an AprilTag's coded center must be recorded as at least one pixel in any photograph in order for the tag to be decoded correctly. Any smaller scale cannot contain enough information to decode the tag. To determine the actual practical resolution limit, a high-quality photograph was taken of an AprilTag mosaic containing 587 individual AprilTags. This was a realistic, but non-ideal image. The image is then resized to varying levels, and the

[1] University of Michigan C/C++ AprilTag Library, available at https://github.com/AprilRobotics/apriltag

AprilTag Pixel Length in Image Pixels	Detection Rate (%)
0.88	0
1.06	10.9
1.25	72.4
1.38	98.9
1.5	100
1.88	100

TABLE 3.1: AprilTag Detection Rate vs Size of Coded Elements in Captured Image

number of image pixels per AprilTag "pixel" element was recorded. Each image was run though an AprilTag detector, and the number of detections was counted. The results, listed in Table 3.1, show that the minimum size of AprilTag pixels, in terms of photographed pixels, must be at least 1.25 image pixels to get a majority of tags detected, and at least 1.5 image pixels for fully-reliable detections. The remainder of this discussion will use the synthetic target of 1.5 image pixels per AprilTag code element pixel as a threshold for usable imagery.

Distance, Marker Size, Resolution, and Field-of-View

These four variables inter-relate to determine the relationship between image pixels and marker pixels, given other optimal parameters.

This discussion assumes a pinhole camera model. Assume that the camera is aimed at plane normal to the camera's viewing direction, and that the plane is distance d from the camera's focal point at its closest point. Given the camera's horizontal and vertical field-of-view parameters f_h and f_v, and horizontal and vertical pixel resolutions r_h and r_v, the physical area captured by one image sensor pixel can be calculated. The AprilTag library has an optional "quad_decimation" setting, ω, which reduces image resolution by the given factor in order to speed execution. These quantities and settings can be combined with the information in Table 3.1 to design a suitable-sized AprilTag for a given camera configuration, viewing distance, and detectability goal. Fig. 3.7 illustrates this geometry for one dimension of an image sensor. The rays cast by two adjacent pixels n and $n+1$ intersect the target plane at coordinates X_n and X_{n+1}.

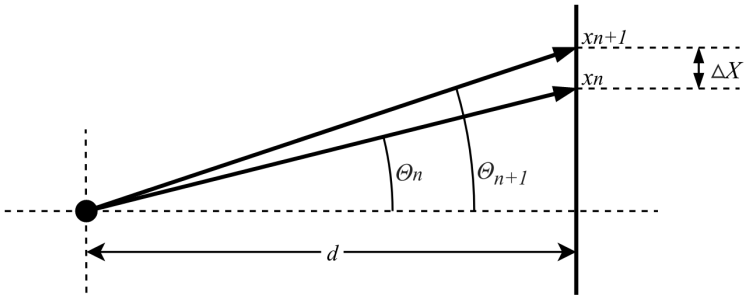

FIGURE 3.7: Relationship between image sensor pixel size and minimum resolvable area on a plane

The rays' angles relative to the lens center can be found by interpolating its pixel coordinates across the camera's field-of-view:

$$\begin{aligned} \theta_{h,n} &= f_h \cdot (n_h/(r_h/\omega) - 0.5) \\ \theta_{v,n} &= f_v \cdot (n_v/(r_v/\omega) - 0.5) \end{aligned} \tag{3.1}$$

Next, the intersection with the target plane can be found by projecting rays with the angles $\theta_{h,n}$ and $\theta_{v,n}$ and the given distance to the plane:

$$\begin{aligned} X_{h,n} &= d \cdot tan(\theta_{h,n}) \\ X_{v,n} &= d \cdot tan(\theta_{v,n}) \end{aligned} \tag{3.2}$$

By inferring that the angle between two adjacent pixels is $\triangle\theta = \theta_{n+1} - \theta_n$, one can find the distance $\triangle X = X_{n+1} - X_n$:

$$\begin{aligned} \triangle X_{h,n} &= d(tan(\theta_{h,n} + \triangle\theta_h) - tan(\theta_{h,n})) \\ \triangle X_{v,n} &= d(tan(\theta_{v,n} + \triangle\theta_v) - tan(\theta_{v,n})) \end{aligned} \tag{3.3}$$

Keeping in mind the practical-application focus of this research, an ideal result would be the identification of a "rule of thumb" formula for estimation. The formula should be easily manipulable, such that any variable can be solved for, given values or ranges for the other variables. Towards this goal, note that for input angles in the range of typical camera fields-of-view (-40° to 40°), the tangent function is approximately linear, so that $tan(\theta)$ can be approximated by θ.

$$\begin{aligned} \triangle X_{h,n} &\approx d \cdot \triangle\theta_h \\ \triangle X_{v,n} &\approx d \cdot \triangle\theta_v \end{aligned} \tag{3.4}$$

Finally, this can be translated back to a function of the camera parameters f and r. For convenience, f will now be given in degrees rather than radians and labelled F, and the equation includes a conversion factor to compensate:

$$\begin{aligned} \triangle X_{h,n} &\approx \frac{1}{57} \cdot d \cdot \frac{F_h}{(r_h/\omega)} \\ \triangle X_{v,n} &\approx \frac{1}{57} \cdot d \cdot \frac{F_v}{(r_v/\omega)} \end{aligned} \tag{3.5}$$

Viewing Angle

As an AprilTag targets rotates away from the normal axis (or equivalently the camera rotates around the target while facing it) it becomes visually compressed, such that horizontal or vertical coordinates are scaled by the cosine of the angle between the target plane and the image sensor plane, which are labelled as a_h and a_v. The pixel resolution is proportional to the viewing angle according to these formula:

$$\begin{aligned} \triangle X_{h,n} &\propto cos(a_h) \\ \triangle X_{v,n} &\propto cos(a_v) \end{aligned} \tag{3.6}$$

Combined with Equation 3.5, a formula for the approximate projected size of one image pixel on a distant surface at some angle of rotation can be developed:

$$\begin{aligned} \triangle X_{h,n} &\approx \frac{1}{57} \cdot d \cdot \frac{F_h}{(r_h/\omega)} \cdot cos(a_h) \\ \triangle X_{v,n} &\approx \frac{1}{57} \cdot d \cdot \frac{F_v}{(r_v/\omega)} \cdot cos(a_v) \end{aligned} \tag{3.7}$$

When combined with the detectability data in Table 3.1, this equation can be used to determine the minimum size AprilTag necessary to meet a specified detectability target at various range / viewing angle combinations.

3.3 Fixed Map and Fixed Navigation Points

3.3.1 The Fixed Map

The fixed map in this system is meant to define the boundaries of safe operation for the drone fleet, as well as the location of navigation aids. The map is maintained in three dimensions, since drones use three degrees of freedom when moving. Simplicity, rather than exhaustive detail, is an asset for this map. This map is the primary source of on-screen location information for the drone operator, and also defines the obstacles used in first-pass path planning as the drones navigate.

Data Structures

For this system, maps consist of *boundaries*, *objects*, *markers*, and a *global reference point*.

Boundaries mark out the extents of the environment - the outer limits of where navigation can be considered. These become constraints limits on path planning, and help the GUI to determine an appropriate scale and initial position. The map boundaries are defined as a floor and ceiling level, and an arbitrarily-shaped polygon of vertical walls.

The *global reference point* defines how map coordinates (which are in millimeters relative to a local datum) are transformed into world coordinates - latitude, longitude, and heading. The global reference point is defined as the latitude, longitude, and heading of (0, 0, 0) in map space. Most parts of the mapping and navigation system do calculations in the internal coordinate system (millimeters right, down, and forward of the origin), but outputs to the flight controller and GUI are transformed into world coordinates. Note that this does require the flight controller to work in sufficient resolution to accept position commands in terms of latitude and longitude without degrading their accuracy beyond acceptable levels for tight-quarters navigation. Essentially they must handle latitude and longitude as double-precision floats.

Objects are the main structural element of the map. Walls, tables, and doorways underhangs are all examples of objects. Listing 3.1 shows a typical object record. There are some simplifying assumptions contained in the Fixed Map system: a flat floor and ceiling are assumed (though objects can protrude above the floor or below

the ceiling). It is also assumed that objects have vertical sides and horizontal tops and bottoms. Objects can have arbitrarily-shaped borders, as long as those borders are made of vertical planes. These compromises simplify many aspects of operation, from the development of an operator-focused GUI that can visualize the map without excessive detail, to simplification of the geometric algorithms used in the drone, such as tests for intersection between vectors and solids.

LISTING 3.1: An Object in the Fixed Map System

```
{
    "name": "diningroom_livingroom_wall_underhang",
    "bottom": -2000,
    "top": -2760,
    "corners": [
        { "r":  4284, "f":  3400 },
        { "r":  4284, "f":   300 },
        { "r":  4434, "f":   300 },
        { "r":  4434, "f":  3400 }
    ]
}
```

Markers are the essential reference points used in the localization system described in this document. Markers are distinguished by their ID value, and must have known coordinates, size, and rotation. Accordingly the map contains a list of all markers with these attributes, as well as a plain-text comment.

LISTING 3.2: Sample Complete Map Definition

```
{
  "name": "Andys Apartment",

  "floor": 0,
  "ceiling": -2760,

  "border": [
    {"r": -10000, "f": -4500},
    {"r": -10000, "f":  4500},
    {"r":  10000, "f":  4500},
    {"r":  10000, "f": -4500}
  ],

  "origin_coordinates": {
    "latitude": 37,
    "longitude": -78,
    "heading": -90
  },

  "marker_type": "AprilTag",

  "markers": [
    {
      "size": 170,
      "value": 30,
      "r": 1662,
      "d": -1204,
      "f": 3740,
      "hdg": 180,
      "pitch": 0,
      "comment": "Dining Room Outer Wall Lower Left"
    }
  ],

  "solid_vertical_objects": [
    {
      "name": "main_outer_wall_a",
      "bottom": 0,
      "top": -2760,
      "corners": [
        { "r": -8000, "f": -3219 },
        { "r": -8000, "f": -3369 },
        { "r":  8330, "f": -3369 },
        { "r":  8330, "f": -3219 }
      ]
    },
  ]
}
```

Creating the Map

The maps used during the development of this system were generated "by hand", by choosing a reference point and then using measuring tapes and distance-finder devices to measure key distances and compute the coordinates of corners, walls, and other objects. For small environments, this method has good results and can be done on the scale of hours. For larger environments, this would become time-prohibitive, and the accumulated drift from stacking measurements on top of measurements could become significant.

It is expected that in the future, tools would be developed to import most map features (boundaries and objects) from an existing architectural document, such as a building information model in AutoCAD. It is important that these tools focus on capturing the essential structure of an environment, without generating excessive detail. As is discussed in the Navigation Points section below, the Fixed Map is not merely cosmetic, but drives the generation of potential navigation routes. An extremely detailed map leads to an extremely complex set of navigation solutions, and will inhibit real-time performance of the system.

3.3.2 Navigation Points

The Fixed Map, being known in advance of any flights, can be pre-processed in ways that reduce the in-flight workload of the system. One application of this prior knowledge is the generation of assets for the path planning algorithm, which is an hierarchical implementation of A* [43] with a non-uniform heirachy. This algorithm, which finds the optimal route between a source and destination point, does not work in open space, but requires a set of points to be given (an undirected graph). The resulting path is the optimal traversal of that graph, given certain weights for each link. In this manner, A* is able to navigate around obstacles, simply due to the absence of graph points within those obstacles.

This graph can be generated using only the information in the Fixed Map, and thus can be generated out-of-band from in-flight navigation decisions. A naive plan would be to quantize all open space, and generate points for each open "cell" of the environment. This quickly becomes unworkable though, due to the overwhelming volume of points being generated where point quantity is related to the third power

of inverse linear point density. (For example, at 1 point per linear foot, a 10'x10'x10' room would have 1000 points. If the linear density is doubled to 2 points per linear foot, that same room will contain 8000 points.) Besides the storage considerations of such a system, it must be noted that the algorithmic complexity of A* is exponential with the depth of solution, which will increase as the number of points between source and destination increases.

A better approach is needed. If "optimal navigation" is defined to mean the shortest safe route, then only certain points in space are ever part of an optimal navigation solution. Corners, specifically, define most useful navigation points. A route that rounds a corner as tightly as possible is typically better than a route that given wide berth to a corner. In a two-dimensional horizontal space, this would be a complete description of the problem and method of finding the solution. However in three-dimensional space for this application, the problem becomes more complex. It's not simply a matter of changing the math from 2D to 3D. As a counter example, consider that a drone is hovering at some altitude, and wants to move to some distant point at the same altitude. Assume that all obstacles are infinite in the vertical direction, so that they cannot be traversed by going above or below them. Intuitively, the ideal path is the 2D navigational solution, executed at the current altitude, since any altitude change only adds distance to the trip, with no possible routing benefit. This is problematic when applied to the goal of creating a minimally-sized graph of navigation points however, since any altitude may be ideal for any point, once again leading to an infinite number of points.

After defining the set of navigation points, one can determine the edges between them. Simply put, there is allowed to be an edge between two points if there is nothing blocking it. However the execution of this idea can be computationally intensive, as it requires computing collisions between the edge and all world objects in 3D space. This makes it another task ripe for pre-computation and removal from the real-time pipeline.

Given that there is a need for a pre-generated set of navigation points and edges, tailored for consumption by the A* path-finding algorithm, and working within the safety concerns of real-world drone navigation, there exists an excellent problem-solving opportunity. The proceeding sections describe the solution taken in this

work.

Problem Statement for the Navigation System

Create an undirected graph of points in 3D space, designed for consumption by the A* path-finding algorithm. The graph should have the following features:

- All potentially useful points for navigation must be included
- Points that are never be part of an ideal navigation path should not be included
- Edges between all points must be created, when those edges don't intersect with the environment or violate distance restrictions.
- The system must apply user-definable horizontal and vertical safety buffers around objects. No points or edges are allowed within these buffer spaces.
- The system must account for the drone's limited area of sensory awareness, and ensure that while navigating the drones next point is always within it's area of awareness.
- The system must minimize the number of points produced.

Additionally, a second phase to this system will be described subsequently, wherein edges that are valid within the Fixed Map, but are determined through the drone's sensing to be obstructed are removed from the graph.

Phase 1: Generate Points from Corners

This phase is simplified by the assumption that all objects have vertical walls, and a flat top and bottom. Given this simplifying assumption, one can look down upon the world as a two-dimensional drawing and handle the corners in only two dimensions. First, the objects are expanded by the user's desired buffer zone size, to create a virtual copy of the object that is larger, and has the buffer zone built into it. Two copies of the expanded object are created - one expanded by exactly 100% of the user's specified buffer size, and one by only 99%. The latter will be useful later, when calculating intersections between edges and objects. Both of these sets of expanded objects are cached for efficient re-use. The process of expanding the objects works on the assumption that all walls should be expanded outward by the given amount, and new corners should be created where walls intersect. This is illustrated in Fig.

3.8. A prior method was tested where corners were expanded and then walls drawn between them, but this proved inaccurate and depended on the angle of the corner.

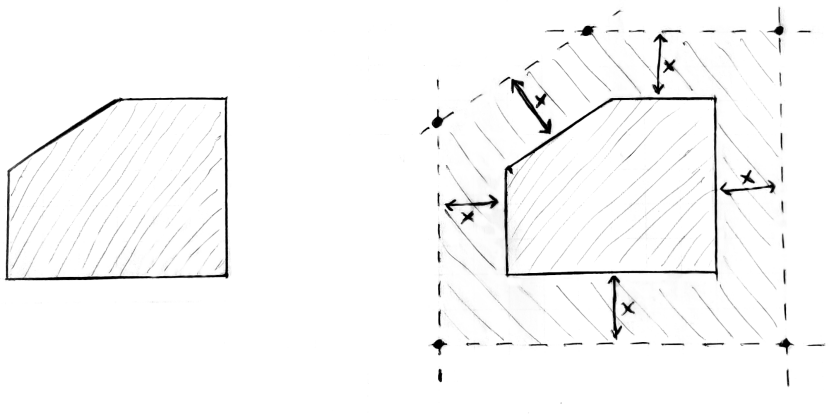

FIGURE 3.8: Object expansion via constant distance from edges.

Vertically, the objects top and bottom are also expanded, by adding the user's vertical expansion amount to the top and bottom coordinates. The actual points are created at the upper and lower bounds of the object, at every corner.

Next comes an adjustment sub-phase. Points that are above the ceiling or below the floor are pushed in-bounds. Points that are out-of-bounds globally are removed.

Finally, there is a filtering sub-phase. Points that are inside of other objects should be immediately removed (since these objects are expanded from the physical objects, it's possible for them to overlap and intersect. Similarly, points that are separated from their source object by another object (for example, a source object near a thin wall may projects points onto the opposite side of the wall) should be removed as well. Here the slightly-less-expanded, 99% objects are used as a filter.

Phase 2: Split Long Paths

Though the current set of points is suitable for navigation in general, an extra step is taken here to accommodate the drone's limited area of awareness. This area of awareness, which is derived from the range of the drone's high-resolution depth sensors, is the area in which the drone maintains detailed knowledge of the world, such that it can safely navigate through unplanned obstacles towards a waypoint. Accordingly each waypoint must be within the scope of the drone's area of awareness as it moves.

In this phase of navigation point generation, potentially useful but too-long links between navigation points need to be identified. When found, those edges are subdivide into segments compatible with the drone's navigation. Furthermore, since this situation tends to occur in clusters (for example a hallway with three points at either end of it has nine similar edges, all generating similar sub-divisions), the operation is performed iteratively, such that extra points created early in the process can be used to "solve" the long-edge problem for later sets of points, and prevent excessive point creation. This does have the side effect of making the final point cloud dependant on the ordering of points, although changes due to point order should be minor.

Phase 3: Remove Blocked Points

In this final stage of point generation, there is an opportunity to remove points that are known to be unavailable at the moment. If occupancy data from the drone exists indicating that the position in space of any navigation point is currently occupied (despite being apparently available on the world map), then that point should be removed from the set.

This implies a minimum requirement for the accuracy of the drone's sensing systems. Since navigation points are by definition created near objects, the drone's sensing system must be sufficiently accurate when detecting those same objects as to not misplace the object onto the navigation point.

Blocked Point Removal is implemented using the expanded objects from Phase 1, so that a point is considered blocked no only if it actually overlaps a physical object, but also if it is within the safety buffer zone around a physical object. Fig. 3.9 illustrates blocked-point removal.

Phase 4: Generate Edges

The preceding phases have culminated in the production of a set of all navigation points that could possibly be of some utility for A*-based navigation within the defined space. Furthermore, it minimizes the number of points, which is critical for performance, by several techniques including the removal of temporarily or permanently unsuitable points.

To further facilitate the navigation algorithm, the "neighborhood" of each point can be pre-calculated, which removes another computationally-expensive process

from the real-time workflow. The neighborhood of each point is the set of points directly-reachable from that point. It can be seen as the menu of immediate options available for navigation from that point. Prototypical A* implementations often show navigation as existing on a two-dimensional grid (such as the pixels on a screen), and the neighborhood of any pixel is the eight pixels surrounding it. Such a design is intuitive and neighbors are easy to identify, but is unsuitable for large-scale, multi-dimensional navigation due to the inefficiently discussed above. By removing unnecessary points, real-time performance of large-area navigation is enabled, however a side-effect is that the calculation of the neighbors of each points becomes more intricate.

Neighbors are defined as points that can be moved to from a point, without requiring additional path planning. They should be within the drone's working radius ("area of awareness"), and they should have line-of-sight to the source point. When visualized on a map, links between a point and its neighbors should always travel through open space, never infringing on objects or the buffer zones around them.

Solving this problem requires the ability to compute the intersection (or lack thereof) between a line segment and a set of plane segments in three-dimensional space. A point can be considered a neighbor if it is both a) within range and b) the line segment to that point does not intersect any face of any expanded object. The mathematical solution to this problem is well-known and beyond the scope of this paper, and can also be safely avoided through the use of a suitable 3D geometry library. Alternatively, one can take the solution employed by the author in his prototype navigation system, and take advantage of the restriction previously imposed on the system with respect to fully three-dimensional geometry to reduce the intersection problem to two-dimensional intersections plus height checks.

Phase 5: Remove Blocked Edges

As in Phase 3, there is an opportunity to inject near-real-time data into the system. This is in fact an essential part of the navigation workflow, to ensure that blocked paths are eventually removed from the system, and that reopened paths are eventually added.

As a reminder of the bigger picture of operation, the drone navigation is a two-part process: first the drone uses the permanent world map to quickly produce a

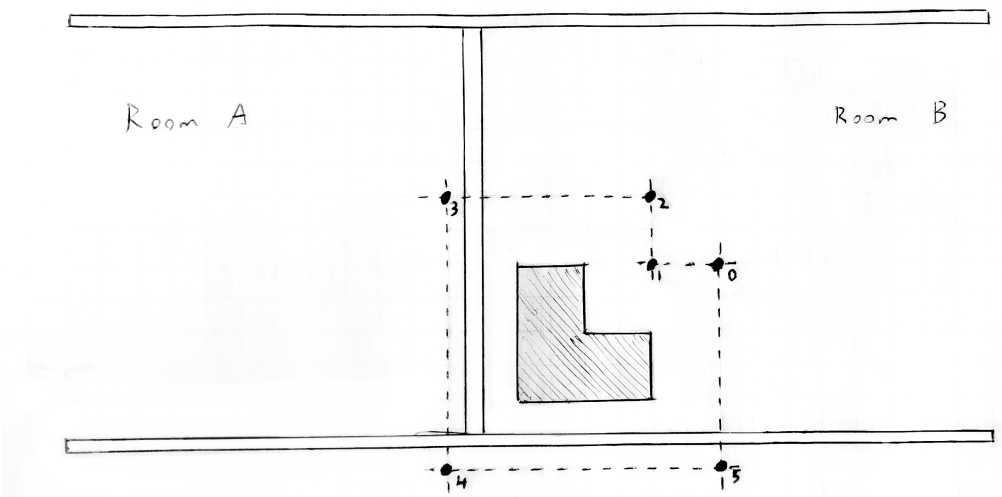

FIGURE 3.9: Example of Points that are in open space, but have crossed a solid object (wall) to get there, and thus are to be removed

rough plan to reach its destination. This produces a series of waypoints. Second, the drone uses its sensory system to map out obstructions within its area of awareness, and map out a detailed route to each waypoint. This process can be re-used however, to map out detailed routes between any pair of points within the area of awareness, based on real-world obstructions as detected by the drone's sensory equipment.

In this phase of map generation, advantage is taken of this capability to determine if the route between each possible pair of points (which is valid according to the predefined map) can actually be executed according to the most recent sensory data. The exact path of execution is not necessary, but if no path is possible, then the world map should exclude that edge and not make the drone attempt to traverse it. Failure to do so make result in a "stuck" drone, constantly trying different paths to get to a waypoint that cannot be reached.

To execute this phase, the set of edges that are within the drone's area of awareness are identified. (Other edges retain their previous passable / not passable state, since there is no new information available to update this state.) The system then simply iterates through each edge, running an A* navigation from one node of the edge to the other. If navigation fails, the edge state is changed to impassible, and if navigation succeeds, the edge state is changed to passable. Impassible edges are removed from the graph, such as in the example of Fig. 3.10.

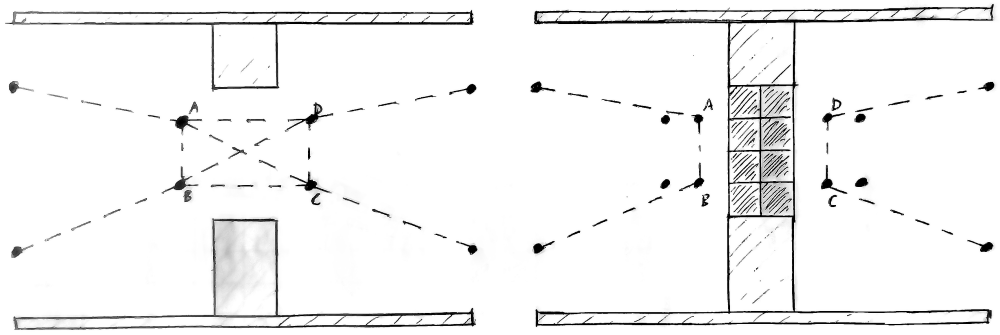

FIGURE 3.10: Example of Fixed-Map Navigation Edges Removed due to Sensed Obstacles

Phase 6: Swap Buffers

The graph is finally complete, and the double-buffering system can swap the new data into service. (Recall that the double-buffering system prevents programming errors such as iterating over changing data structures which the graph is being regenerated).

3.3.3 Online Operation

Although the preceding explanation of Fixed-Map Navigation Point Generation was described as a pre-processing step, the actual implementation is more complex. It can be seen that the inputs to this system are the Fixed Map, which is static, but also sensory data from the drone, which updates based on the drone's location, orientation, and time. Therefore to continuously integrate new sensory information into the system and generate an optimal set of navigation points, this algorithm must be run in a loop. At the same time, this potentially slow Navigation Point Generation algorithm should not impede the performance of the 2nd-Stage navigation system, which is more critical to second-by-second navigation of the drone.

To balance these goals, multi-threading and double-buffering are used. The Fixed Map Navigation Point generation algorithm is run in a private thread, generating a private data set. When finished, that dataset is published and the process restarts with a new private dataset. Meanwhile the public dataset is available for consumption by other parts of the system. The public dataset is protected by mutexes, to ensure that it is updated in a way that isn't harmful to any consumers of

the dataset.

As a result of this procedure, a graph of navigation points optimized for A* navigation is always available to the navigation system, and updates itself in the presence of newly discovered obstructions or cleared spaces within a matter of seconds. The drone's navigation system is frequently re-plotting its route against this map, so updates can be turned into actionable movements immediately and automatically.

Finally, the GUI and communications system developed for this project includes the display of navigation points and edges, either in a 2D top-down view, or in a 3D space with arbitrary camera positioning. These tools have been invaluable in developing and debugging this technology. (As a side note, the project starts to touch upon the idea of "too much data" here, where the visualizations can be more overwhelming than illuminating. As a developer of such a system, it is essential to be able to quickly build tools to help interpret and filter data. For example, unit and functional tests can be created to impose repeatable test conditions upon a system.)

3.4 State Estimation

The need for an awareness of the position, velocity, and attitude of the aircraft shouldn't require much explanation. How can the aircraft steer itself towards its destination if it doesn't know which way it is facing? In an ideal world, with perfectly accurate sensors with zero latency and infinite update rates, this would be as simple as reading the sensors. The word "estimation" wouldn't be required, it would simply be a state measurement.

While waiting for these perfect sensors to be invented, one can make due with a synthetic estimate of state, derived from multiple sensors. In doing so, one can choose to optimize for certain attributes, at the expense of others. Latency and noise rejection are often at odds with each other, so one can choose to favor a highly-responsive signal when useful and a low-noise signal other times. For example, the drone's position control function relies on the velocity estimates for moment-to-moment feedback. A high-latency velocity estimate makes strong position control impossible, while a low-latency, high noise velocity signal is still usable. Additionally, tradeoffs such as internal consistently vs. purpose-optimized individual outputs are possible.

This section describes the state estimator developed during the performance of this Visual Navigation System project. The state estimator must produce estimates of the aircraft's 3D position, velocity, acceleration, and attitude. The system should be capable of running as quickly as it's fastest input. The final uncertainty of each output state should be known, as should the system latency.

From a birds-eye view, the State Estimator outputs are connected to the Aries Flight Control, where they are it's position, velocity, and heading data sources. Internally, the state estimator runs in a Predict-Update-Feedback loop, as illustrated in Fig. 3.11.

When designing the state estimator, physical congruity of state components can also be balanced against the desirable characteristics of individual signals. In the physical world, acceleration, velocity, and position are integrals and derivatives of each other. However as separate components of a state estimate, this requirement can be relaxed.

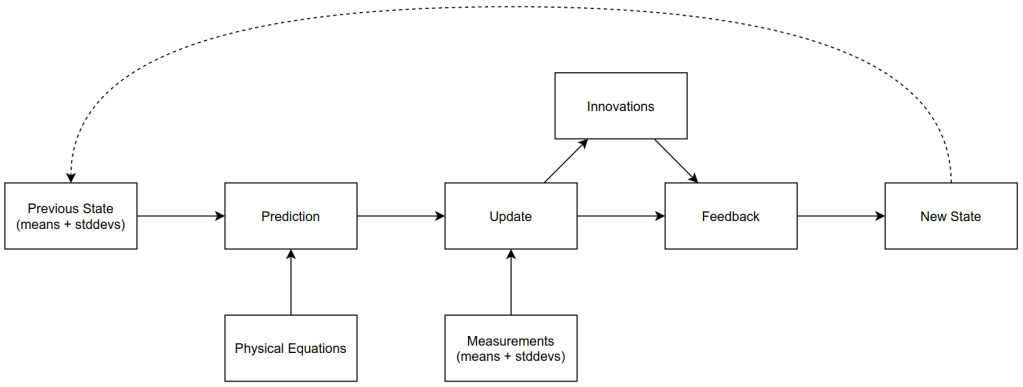

FIGURE 3.11: Visual Navigation State Estimator Flowchart

The state estimator described herein is a variant of the Wiener Filter documented in Section 2.1.2. This filter method was chosen for its simplicity and reliability, especially its lack of failure modes present in Kalman filters. It was also chosen for its speed of execution, especially when compared to the Particle filter. Finally, although it is understood to be less optimal than either the Kalman or Particle filters can be when operating at their best, it is more than sufficient for the task at hand. Its implementation, which focuses on identifies the statistical properties of the input signals, is in line with the educational goals of this project.

Consider the system's state estimator which estimates position, velocity, and acceleration. Among the data sources available to it are position measurements from the AprilTag system. A situation could arise where the drone is travelling with a medium-accuracy position estimate, until at some point a large, near-range AprilTag comes into view. Given this high-quality data source, position measurements also become more accurate, but may jump suddenly as the position estimate snaps into line with the better measurements. Physically, velocity is always the derivative of position, but in this situation it would be incorrect to update the velocity estimate by the time-derivative of the position estimate.

It is critical to understand the use-cases for elements of a state estimate when considering a case like this. A multi-rotor drone has a strong reliance on velocity estimates, because without high-speed, low-latency velocity data, the drone cannot maintain horizontal or vertical stability. Velocity control is critical and depends on a low-latency data stream. Position control on the other hand is further removed from

the realm of basic flight stability. With a strong velocity controller to lean on, a weak, slow, or noisy position signal can still be effective. Therefore this application favors a partial decoupling of velocity from position, where velocity can be integrated into position, but position is not used to correct velocity.

3.4.1 Step 0: Initialization

The state estimator's core data structure is an array of state variables, represented as Gaussian distributions. Each index of the array represents one dimension of state: X Position, Y Position, Z Position, X Velocity, etc. The uncertainties of each state are initialized to a large number, and their mean is initially undefined.

As the system starts up, it waits for all data streams to come online, and for their outputs to stabilize. Once deemed to be healthy, the system grabs position, velocity, and acceleration samples directly from the sensors (AprilTag positions, ZED Camera velocities, and Aries FCS accelerations), and uses them as the initial means for the state Gaussians.

3.4.2 Step 1: Time Synchronization

Sensor fusion can suffer from the issue of varying and variable time latencies between sensors. Intuitively it is assumed while fusing data that all sensors gather different information about the same basic system - the same aircraft at the same moment in time. In reality, this is rarely the case, as each sensor and it's varied communication and processing pathways create latency that is specific to the individual sensor. If used without correction, the system would be fusing information about different points in time together as if they were all synchronized.

This problem can be addressed by creating an artificial time horizon where data exists for all sensors, and then solving the state estimate for that point in time. Since this yields a past state estimate, rather than a present state estimate, data samples covering the period of time between the time horizon and the current time can be replayed upon the delayed state estimate to "play it forward" to the present time.

The Visual Navigation System contains a "Delay Solver" component that (when possible) automatically determines the relative delay between two data streams. The Delay Solver works as follows:

- First, two data streams are passed through the solver. As each new sample arrives, its derivative is placed into circular buffer, along with a current timestamp. Each buffer holds five seconds of data. The derivative was chosen to eliminate steady-state or low-frequency information from the data.
- In a separate thread, the Delay Solver periodically starts its update cycle. Here, a series of processes happen:
    1. The window of time common to both buffers is identified.
    2. Data in both buffers is upsampled to millisecond resolution via interpolation.
    3. A cross-correlation is run on the two time-aligned, upsampled buffers
    4. The peak of the cross-correlation output, and it's associated time interval is discovered.
    5. Quality checks are done to determine the information content of that peak. Not all data can produce useful results, as is the case when the aircraft is sitting motionless.
    6. If found to be usable, the cross-correlation delay is applied to the final delay value, via a weighted moving average.

The Visual Navigation System uses instances of this delay solver to identify the relative delays of the ZED velocity data and AprilTag position data relative to the Aries acceleration data, which is assumed to have the lowest latency.

The state estimator maintains circular buffers of timestamped recent data samples from all sources. Given the latencies calculated above, it can determine a suitable time horizon for state estimation by finding the maximum of the individual sensor latencies and subtracting that from the present time. The state estimator then computes the state estimate at that point in time, using the correct samples pulled out of the sample buffers.

The "play it forward" functionality described above is not implemented, but can safely be ignored in this particular application, since the state estimator outputs to the Aries FCS, which has similar functionality built into it.

3.4.3 Step 2: Filter Prediction

The state estimator runs in a predict-update-backpropagate loop, similar to a Kalman filter with no covarience matrix.

In the prediction phase, the state estimator does two things:

- Account for the passage of time by increasing the variances on all signals
- Update states by applying a physical model upon the last state

Note that throughout this section state variables, inputs, and outputs are assumed to be probability density functions (PDFs), containing a mean and variance. Constant values are notated as a (mean, variance) tuple. Multiplication and addition of PDFs in the formulae below follows definition of PDF multiplication and addition:

$$a \cdot b := \begin{cases} \mu := \frac{\mu_a \sigma_b^2 + \mu_b \sigma_a^2}{\sigma_b^2 + \sigma_a^2} \\ \sigma^2 := \frac{\sigma_a^2 \sigma_b^2}{\sigma_a^2 + \sigma_b^2} \end{cases}$$

$$a + b := \begin{cases} \mu := \mu_a + \mu_b \\ \sigma^2 := \sigma_a^2 + \sigma_b^2 \end{cases}$$

To account for the passage of time, a defined rate of variance increase, notated as n_i is applied to the current state variables, which are indicated as x_i:

$$x_{i,step1} = x_{i,last} \cdot (1, n_i dt) \tag{3.8}$$

The physical model is then added to the previous equation by assuming constant acceleration, and then integrating acceleration into velocity and velocity into position:

$$x_{a,step2} = x_{a,step1} \tag{3.9}$$

$$x_{v,step2} = x_{v,step1} + x_{a,last} \cdot dt \tag{3.10}$$

$$x_{p,step2} = x_{p,step1} + x_{v,last} \cdot dt + \frac{1}{2} x_{a,last} \cdot dt^2 \tag{3.11}$$

3.4.4 Step 3: Filter Update

Next, sensory inputs are merged into the current state estimates. Note that the gaussian products used here operate as a variable-frequency complimentary filter.

$$x_{a,step3} = x_{a,step2} \cdot i_a \tag{3.12}$$

$$x_{v,step3} = x_{v,step2} \cdot i_v \tag{3.13}$$

$$x_{p,step3} = x_{p,step2} \cdot i_p \tag{3.14}$$

Note the condition that checks to make sure an appropriate sample can be found. It is often the case that there is no new measurement data available, since this system gives each sensor and software component the freedom to run at differing update rates. Additionally sensors can experience data dropouts, such as the position system when no visual markers are in sight. Under these conditions the Prediction stage of the filter continues to propagate the state estimate forward in time, while using whatever data is available to improve it.

At this point Innovations (defined as the change in mean value caused by measurement updates) can be calculated, as shown in Equations 3.16 and 3.17. Innovations are a basic health indicator, as large-magnitude values can indicate disagreement between sensors. They are also used in the feedback phase, below.

3.4.5 Step 4: Feedback

Finally, the amount of correction generated by the previous step is measured and pushed through the physical model in the reverse direction. Keeping in mind that all inputs are discrete and differently-timed, it is frequently the same that some states

are not updated while others are. This feedback phase allows any new information to affect all states appropriately. In practice a strength coefficient C_s was added to this process, allowing it to be detuned for testing purposes.

$$i_v = x_{v,step3} - x_{v,last} \tag{3.15}$$

$$i_p = x_{p,step3} - x_{p,last} \tag{3.16}$$

$$x_{a,step4} = x_{a,step3} - i_v C_{s,a} \tag{3.17}$$

$$x_{v,step4} = x_{v,step3} - i_p C_{s,v} \tag{3.18}$$

These steps culminate in a state estimator capable of creating a high-quality estimate of all states even in the presence of input sources of differing data quality (both intrinsically and over time) and differing update rates. In the event of dropouts of individual sensors, all states will continuing to be updated using the remaining data sources, though their variances may grow unacceptably large over time.

3.5 Occupancy Grid and Occupancy Nav Points

3.5.1 Resource-Constrained

The sensory system is responsible for building a high-resolution map of the immediate environment, so that the navigation system can decide upon a safe route to its destination, even in the presence of unforeseen obstacles. It uses the raw sensory information - sets of depth measurements - to produce a Minecraft-like voxel representation of the world. Finally, that geometry is processed to produce a graph of possible path segments suitable for traversing the obstacles while maintaining a safe buffer distance.

No sensors are ideal, so sensor errors and noise must be planned for. Furthermore, the sensors output is generally understood to be relative to the drone's current position and attitude in space, which is also an estimate and subject to various uncertainties.

The culmination of these factors is a subsystem that is algorithmically complex and must deal in probabilities rather than absolutes, but moreover is subject to the hash reality of $O(n^3)$ complexity[2]. This scaling issue will come to dominate the design of this entire subsystem. In this chapter, the subsystem is described, including the optimizations and complexity introductions introduced in order to obtain acceptable performance.

3.5.2 System Workflow

The pipeline can be divided into three major phases: data intake, voxel processing, and graph processing. In the data intake phase, raw data is read from the depth sensor and transformed into world-aligned data with uncertainty. In the voxel processing stage, those measurements are used to update the local occupancy map. Finally, in the graph processing phase, a graph is generated of navigation options optimized for consumption by the navigation system.

Data Intake

[2]The space complexity of a 3D voxel field. Overall time complexity is a function of the constituent voxel field, graph processing, and navigation algorithms complexities, which differ.

This section considers that the main data source is the depth image produced by a ZED Stereo camera. Other data sources, such as RADAR and LIDAR would be handled similarly, and could even be multiplex into a single composite data set. Of course specific details related to the sensor vary.

The ZED Stereo camera produces a video stream of depth images. These images are represented internally as two-dimensional arrays of floating-point numbers, with each number recording the computed distance between the camera and the object seen by that pixel. When combined with the RGB image simultaneously produced by the camera, both the color and distance to the object seen by every pixel of the camera are available. Sample images from the ZED Stereo Camera are displayed in Fig. 3.12. Hopefully it is clear from this description that the depth measurements are limited to the extents of the camera's field-of-view, which varies with resolution but is around 54° vertically and 85° horizontally. Drones such as the Skydio 2 have solved this limited-visibility problem by using several pairs of image sensors, combined with fish-eye lenses to create stereo images with full spherical coverage.

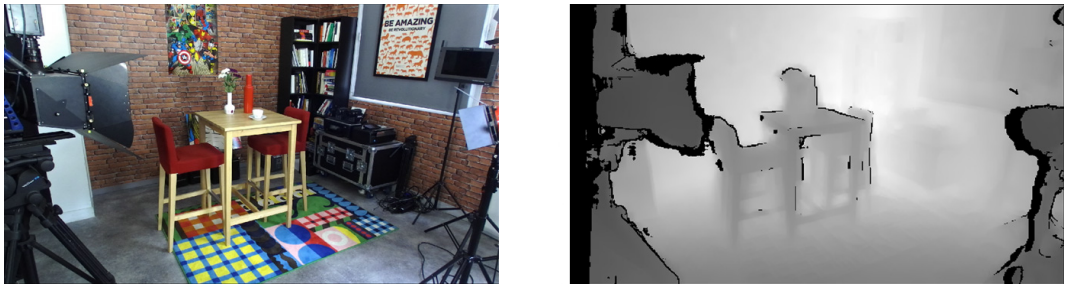

FIGURE 3.12: Sample Image output of the ZED Camera: RGB and Depth (visualized as Greyscale)

It should be noted that these depth images are the product of image processing algorithms rather than direct sensory information, and carry with them a unique set of characteristics. In areas where the light level is above or below the limits of the camera's exposure configuration, and in areas where there is no discernable detail within a group of pixels (textureless, solid colors), the system may fail to produce a result, although this failure is correctly communicated on a per-pixel basis. Furthermore, the system lacks the computational power to do a true per-pixel solution, so it uses undisclosed average and filling algorithms to expand a downsampled set

of solutions to the full image. This process creates significant errors in depth measurements, and that error is non-Gaussian. There is a setting exposed to adjust the strength of this downsampling process. Since the ZED's image processing logic is implemented primarily on the host system, rather than on some camera hardware, its computation performance is directly affected by the host systems abilities. At the same time, it can have a significant impact on the host system's available CPU and GPU resources.

The final depth image stream has a user-selected pixel resolution, user-selected frame rate, and user-selected depth map quality. The test system in this paper uses the following settings, which focuses on higher-quality data, at a speed that is acceptable for real-time operation.

- Resolution: 1280x720 pixels
- Frame Rate: 15 frames per second
- Depth Map Quality: "Ultra"

As a final note about the sensor, this same resolution setting applies to the RGB image, which is used for AprilTag pose estimation. That process is sensitive to camera resolution, so an ideal, but currently unsupported, configuration for the ZED camera would be maximum-resolution visual images, with lower-resolution depth images to support performance requirements.

Voxel Processing

Given this set of depth measurements, and given the aircraft's current position, points can be plotted in 3D space about where solid objects are at the moment. This is done by ray-tracing beams outwards from the camera's image sensor for the measured distance. For example, assume that the drone's position is P_d, the camera's position relative to the drone is P_c, the drone's current attitude is described by the rotation matrix R_d, the vector of a ray leaving the camera at pixel location (x, y) is described by the rotation matrix $R_c(x, y)$, and the measured depth at that pixel coordinate is d. The following equation describes the location of the solid object hit by that ray:

$$P_{object} = P_d + (R_d * P_c) + (R_{c(x,y)} * R_d * d)$$

Over time and repeated measurements, data can be accumulated into a persistent map of the environment. Given the uncertain nature of both the depth field measurements, and the aircraft position that they are related to, these measurements must be distributed to the voxel field probabilistically, and the voxel field must be maintained as Gaussian values (mean plus uncertainty) for each voxel, rather than a binary on/off or a simple average.

After an initial step of downsizing the depth image to reduce the number of points to be processed, the system iterates though each depth sample. For that sample the system distributes the positive probability of there being a solid object at that location, according to the certainty of that sample, and according to the certainty of the aircraft's position and orientation at the moment.

Next, the same process, but with a negative probability is performed on the space *between* the camera and the depth hit, since it is presumably open space for light to travel through. This line segment is sampled into a series of points, each of which are merged into the voxel field as probability of open space. The end result is a voxel field where voxels with a high mean and low uncertainty can be treated as solid objects, and other voxels treated as clear or unknown. A rendering of this system in operation is included as Fig. 3.13.

There is a risk in the process of developing false certainty, due to the repeated integration of samples. For example, while sitting still but acquiring samples at 15 frames per second, the system is going to "see" the same positive and negative inputs multiple times. Care must be taken mathematically to handle these probabilities appropriately. They cannot be multiplied together as Gaussian Probability Density Functions (PDFs), since that equation reduces uncertainty compared to either of the inputs, and thus would create artificial certainty where none should exist.

Local Navigation Area and Drone Movement

The voxel field in this system is a fixed size, to set an upper bound on the resources required to process it. However this is fundamentally incompatible with

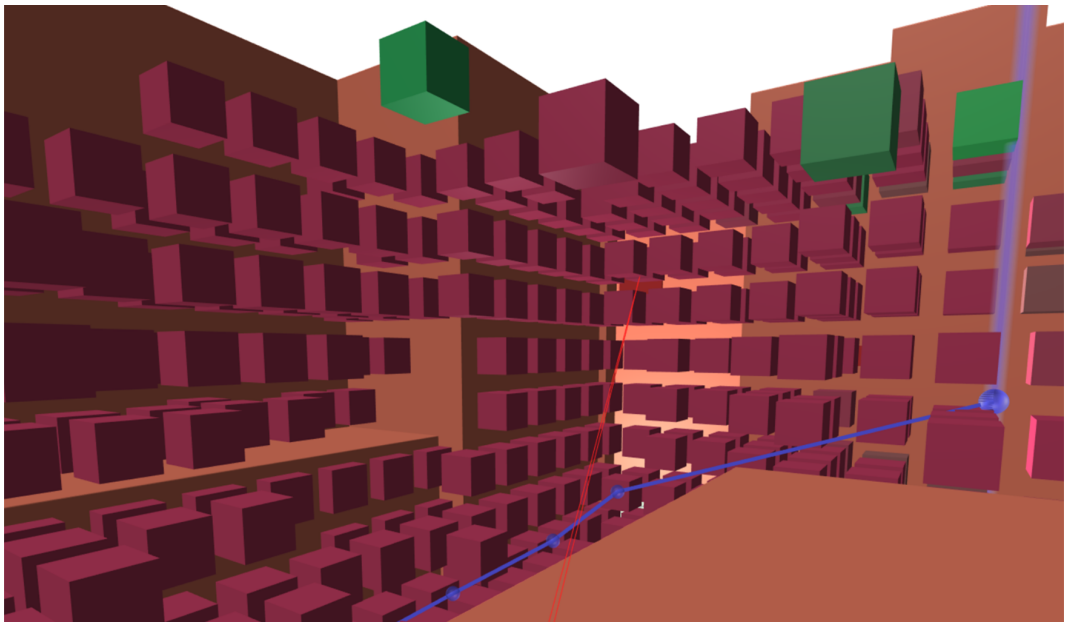

FIGURE 3.13: Sample Voxel Field from Occupancy Grid System)

the goal of large-area navigation, with no hard size limit. This conflict is the motivating factor behind the two-stage navigation approach taken in this system. It was hypothesized that in most cases detected obstacles make only small changes to navigation routes. If that hypothesis holds true, then routes can be generated without knowledge of local obstacles, and then modified on-the-fly as the drone moves through space.

A key idea here is that the local area for which obstacle detection is available can be small, and follows the drone as it moves. To this end, the voxel data is fixed-sized, and moves with the drone. Other stages of the navigation system are designed for compatibility with this concept of dual-scale navigation. For example, the large-scale path planner never generates successive waypoints that are too far apart to fit within the local navigation zone.

This concept does cause the issue of data "falling off the map". As the local navigation zone slides in a direction, voxels on the receeding edge go out of scope and be deleted. These data can be persisted to long-term storage before deletion (effectively expanding the voxel field back to infinite size, though split into accessible and inaccessible components), or they can be simply deleted. Similarly, on the advancing side, voxels can be loaded in or initialized to their default, "no information" value.

Graph Processing

Given that the system now has a map of its surroundings - the voxel map, a set of navigation options can be developed for safely navigating through these surroundings. This is a similar process to the graph generation performed by the Fixed Map system, but tailored for operation with a dense set of voxels.

Graph generation starts by expanding each voxel to include a user-defined "buffer zone". This zone should be sized such that as long as the drone's coordinate position does not contact the expanded voxel, then the drone itself clears the voxel by some safety margin. Remember that the drone has a physical size, so its extents are different from its coordinate position. A reasonable buffer zone size might be the drone's radius plus one foot.

Next, and similar to the fixed-map processing, graph nodes are created at each corner of each expanded voxel. The vertical axis of points is handled differently from the fixed map: in the fixed map z-axis values were simply assigned, and there could be multiple values per point to create navigation options. This occupancy-grid-based system instead locks z-axis values to their computed positions, with the exception of points being very close to the floor or ceiling are pushed further away.

This is followed by a series of culling steps designed to remove points that are for some reason invalid. Points that are simply out-of-bounds compared to the predefined global limits are removed. Points that are below the floor or above the ceiling are removed, and points just in-bounds of the floor and ceiling are moved up or down to create a minimum clearance. Points that are inside, or on the edge of, another expanded voxel are removed. Points that are inside, or on the edge of, any expanded fixed-map object are removed as well. Finally duplicate points (points at the same coordinates) are de-duplicated. At the conclusion of this culling phase, the graph contains all "legal" points of interested, based on the voxel map.

There is however one final culling step, as "legal" doesn't always mean "useful". Points of interest for navigation are generally corners, and this voxel-based graph generation tends to produce many co-linear line segments. As a lower-dimensional example, consider the set of points constructed around the occupancy grid in Fig. 3.14. Points B, C, D, G, H, and I can be seen to superfluous with regard to path generation, by the fact that their removal has no effect on the shape of the pathing

options created. Programatically, they can be identified as points where every neighbor can be paired with a second neighbor, such that the two neighbors are parallel.

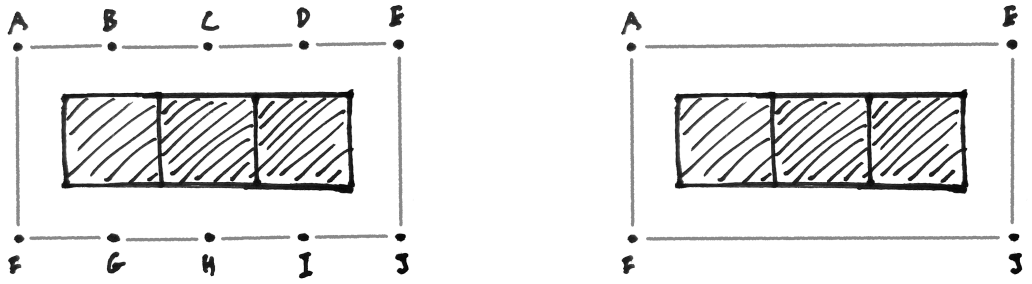

FIGURE 3.14: Example of Point Set Before and After Co-Linear Point Removal

Finally, the edges must be generated between nodes, this is an expensive $O(N^2)$ search, so the preceding efforts to reduce the number of points in the set are vital to gaining acceptable performance. Once again, there are generation and culling steps performed to optimize the final data set. Initially, nodes are joined with an edge as long as they're within range of each other, and the edge would not cross through any expanded voxel, or expanded permanent object. In can be seen though, that from a pathfinding perspective, that this creates redundancy, as it includes both step-by-step hops to get to a destination, as well as longer, direct routes. While these aren't truly identical, if checked for a sufficient level of similarity, the number of edges can be significantly reduced with little effect on ultimate path cost. This has the side-benefit of making the graph cleaner from a visualization standpoint. Therefore, the edges are culled through a process where edges emitting from a node in similar directions are grouped together, and only the shortest edge in the group is kept. This is illustrated in Fig. 3.15.

FIGURE 3.15: Example of Point Set Before and After Similar-Angle Culling

The end result of this system is a minimally-sized set of navigation points that

can be directly consumed by an A* navigation algorithm. Visually, it appears as a web of routes surrounding the obstacles, always separated by a safe distance, such as in Fig. 3.16.

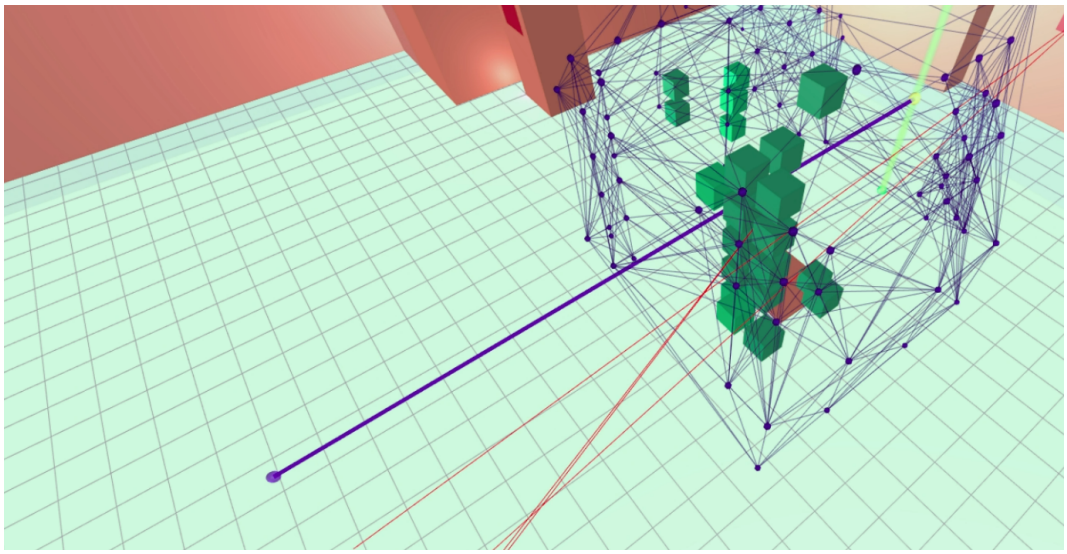

FIGURE 3.16: Sample Navigation Graph (blue) generated from detected occupancy grid (green))

3.6 Anti-Collision System

Commercial aviation uses a sophisticated system of air routes, air traffic control, and flight planning to ensure that aircraft remain safely separated from each other. Compared to any other form of vehicular traffic, it is has a phenomenal track record. This is due in part to the risk-reducing practices that are taken at every step of the way: check-lists and cross-checking are the norm, communications are scripted and designed for maximum effectiveness with minimum overhead, and alternate outcomes are always considered and planned for in advance. These practices have been honed over decades, building upon the experience of generations of pilots and air traffic controllers. And despite all of this, commercial aircraft contain a device called the TCAS which single-handedly supersedes the authority of all planning and all air traffic control personnel. The TCAS ("Traffic Collision Avoidance System") detects threats of imminent collision with nearby aircraft and issues resolutions advising the pilot to either climb or descend in order to mitigate the collision risk. When activated, the pilot *must* follow the TCAS advisories immediately, ignoring air traffic control if necessary to do so.

The TCAS system is interesting because it pairs simplicity and isolation with absolute authority, in an industry that is extremely risk-averse and reliant on checks and balances. Yet this is appropriate, because the industry has recognized that being risk-averse does not mean zero risk, and complexity tends to imply higher and unforeseen risks. As a last-resort safety system, its strength is in it's simplicity and attendant reliability. These are the reason that it is the only system vested with the authority to issue directions that are beyond question.

3.6.1 Collision Detection

The navigation system can benefit from a similarly reliable last-ditch collision avoidance system. Therefore such a system was created and activated, prior to the developed for the full navigation system. The collision-avoidance system simply detects when there is an object in the immediate vicinity of the aircraft, and refuses to allow the aircraft to move forward into that object.

Detection works on the same depth data that informs the obstacle mapping and navigation system. However for this application no map is created, no statistical means are tracked, and in fact there is no stateful data at all. Depth samples are simply placed into bins according to their distance. The lowest two bins (3 to 1.5 feet, and 1.5 to 0 feet) trigger warnings and actions respectively. Bins are also sorted by horizontal position (left-to-right from the point of view of the camera) in order to provide additional feedback to the user about the source of the caution/alarm. On the GUI, proximity warnings are rendered as orange and red boxes covering the areas where objects were detected, as in Fig. 3.17. This system is illustrated in Fig. 3.18.

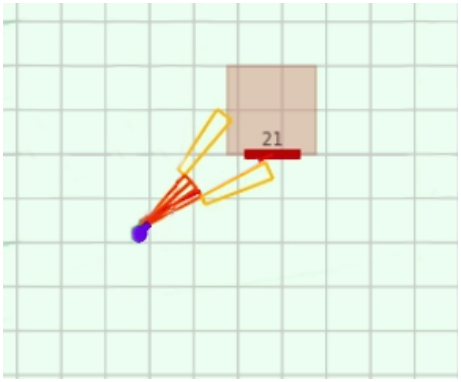

FIGURE 3.17: Anti-Collision indicators on the GUI. The aircraft (blue) is approaching a column (brown), which triggers the emergency proximity action for the center three bins (red), and triggers the warning action for the outer two bins (yellow).

A mask is applied to the depth samples before processing to exclude known-bad samples. Due the the location of the ZED Camera (the source of depth information), the drone's propellers and propeller guards are visible in the camera's field-of-view, and generate erroneous readings. These areas of the image are masked off before processing. This pipeline is able to run easily at the full frame-rate of the system system, providing real-time information to the flight controller.

3.6.2 Aircraft Response

The drone flight controller (VCU "Aries") was modified to accept this proximity data, and if possible to act on it. If this aircraft is in any position-controlled mode of operation (autopilot, or user-driven "Position Hold" mode), then when asked to move forward, the aircraft's target velocity is instead immediately set to zero. This, as an

input to the aircraft's lower-level stabilization systems, causes the aircraft to bank as necessary to arrest its motion, and then continue to make adjustments to maintain a zero velocity. In essence, the aircraft comes to a stop immediately and refuses to move forward. However backwards and sideways movements are still allowed to help the pilot exit the proximity situation.

It should be noted that the aircraft is not commanded to hold or retreat it's position, but merely to attempt to make its velocity zero. Since zero velocity is a target for an imperfect control system, there is the possibility of drift, including drifting towards the obstruction, over time. This compromise was chosen to focus on simplicity, as position control brings another set of control loops and state estimates into action. In the event of a misbehaving position estimation system, which is certainly possible during this research project, the position control will induce motion as it reacts to changing state estimates.

3.6.3 Interaction with Other Systems

As with the TCAS, the Collision Avoidance system is designed to minimize dependencies on other systems, and circumvent the as much of the normal pipeline as possible. It does not participate in the normal navigation or environment mapping systems, but does not inhibit them either. On the flight controller, the velocity override system described above allows the normal navigation systems to communicate and execute as usual, except that their final output is overwritten by the safety values from the collision avoidance system.

As a software component, the Collision Avoidance trigger runs in a separate thread. It's main data source, the ZED Camera driver, transfers data to both the Collision Avoidance System and to the normal Obstacle Mapping systems through copying and semaphores, so that neither consumer blocks the ZED driver from continuing to run. This clean hand-off from thread to thread helps ensure that a stall one one part of the system wont affect unrelated parts.

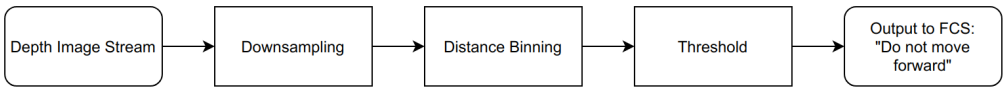

FIGURE 3.18: Collision Avoidance System pipeline

3.7 Commander

3.7.1 Concept

The navigation system described herein is composed of many components, each responsible for specific functionality. Each software component can be visualized as a member of a team with specific expertise. That member should be trusted within their area of expertise, but only in that area. The team should be lead by another member - one who's expertise is in leadership. The leader, or "Commander" to pilfer PX4's software vocabulary, gives orders, awaits results, and then acts upon those results. They see a minimum, but appropriate amount of detail, and have predictable, bounded actions.

3.7.2 Parameters

The Visual Navigation System's commander has one goal: to get the aircraft to its destination. At its disposal are the two mapping systems (architectural and sensory), path generators, the vehicle's current state estimate, and the user's instructions. At its command is the aircraft itself, in the form of movement commands sent directly to the flight control system. The user specifies the desired aircraft state (position, altitude, and heading), and some parameters describing how the command should go about navigating.

As a way of introducing the capabilities of the commander (and thus the system as a whole), let's start by examining the options affecting its operation.

Movement Modes

The Commander has configurable movement modes:

- *Direct:* This mode is "As the crow flies". In this mode the vehicle will simply move directly towards the user's waypoint. No pathfinding attempt is made, and thus no attempts to plan around architectural or sensed obstacles is made. The collision-avoidance system is still engaged, but this mode does not intend to be robust against crashes.

- *Architecture:* This mode uses its pre-loaded map of the environment to plan around known obstacles. It incorporates sensory information only insofar as

routes that were previously seen to be completely blocked will still be treated as such. A configured "buffer zone" is placed around all solid objects, and then paths for the drone are developed that navigate to the destination while avoiding encroaching on these buffer zones.

- *Architecture + Sensing:* This is the normal operation mode. Paths are generated based on buffer zones placed around both pre-mapped objects, and sensed objects. The path is regenerated continuously as the drone moves and discovers new obstacles, so the path can change in mid-flight, but will still eventually reach its destination as long as it is possible to do so.

In general, a multirotor's heading is de-coupled from its direction of motion. This allows the heading to be used in any way that is advantageous while navigating. The test vehicle's heading is strongly coupled to the camera system, so it can be useful to control the camera's heading in flight to accomplish certain tasks, like keeping visual aids in view. Alternatively, it may be useful to align the camera with the direction of travel, to aid in collision avoidance. Given this plurality of use cases, the system allows the user to choose the algorithm used to control the vehicle's heading:

- *Direct:* The heading is what the user specifies, and does not change during motion.

- *Waypoint:* The vehicle will rotate to face its current waypoint as it travels. This maximizes the effectiveness of front-facing collision-detection, and similarly aids in the discovery of new obstacles in the drones immediate path, to be incorporated into its sensed-obstacle map.

- *Nav Aids:* The vehicle will rotate to face the current best option for a navigational aid (AprilTag), where the "best" is defined based on a formula of the vehicle's distance to the marker, the marker's size, and the angle of the vehicle relative to the marker. Some hysteresis is applied to prevent the vehicle from toggling between similar-quality markers.

3.7.3 Control Logic Description

The commander, when operating in anything other than Direct movement mode, is implemented as a state machine. Fig. 3.19 shows the state transition diagram for the Commander. Conceptually, the commander goes in a three-stage loop: Macro Route, Micro Route, Output. It iteratively maintains an overall plan to reach the destination, while generating a detailed plan of its current step.

"Macro" is the term given to the Architectural or Fixed-Map data. This is the pre-loaded, human-created data describing the geometry of the spaces the drone will be operating in. It is considered permanent and unchangeable, except that possible routes can be marked as unavailable temporarily by the sensing system (discussed below). When the Commander enters the Macro_Route state, it asks for a new solution to be generated for navigation from the aircraft's current position to the user-input target coordinates. This request is fielded by the A* pathfinding module, using the Fixed-Map data. It is possible for this request to fail, such as when a request is made to generate a path between two points that are mutually inaccessible. Failure is handled by reverting back to the beginning of the state machine and trying again. This could eventually lead to success, if the vehicle's current position changes, or the unavailable links set changes.

"Micro" is then the term for generating a detailed route to avoid any sensed obstacles. This plan is potentially of much higher fidelity than the macro plan, and is correspondingly more resource-intensive to create. Therefore it is focused only on getting from the aircraft's current location to the first waypoint, as defined by the Macro plan. Extra care is taken that the waypoints generated by the Macro system are never so far apart as to exceed the range of the in-memory sensed-obstacle data, so that it can be safely assumed that this local high-resolution map can be used as an input for a complete A* navigation pass. This Micro step is skipped entirely if operating in the Architecture-only mode.

Since this is an iterative system, it only ever needs to reach the "first" waypoint. As it approaches that waypoint, a new macro route will be created using the aircraft's current position as the start-point, and will generate a new "first" waypoint closer to the destination.

With regards to the ongoing research into A* variants (Section 2.3.1), and in particular hierarchical A* variants, this system is most accurately described as a hierarchical A* variant with a non-uniform hierarchy. With most published hierarchical A* implementations, the hierarchy and multi-level search algorithm are internal to the pathfinding system, making the hierarchy invisible from an external perspective. This system, on the other hand, explicitly manages the two levels of hierarchy, and treats each one uniquely. Within a given level, the A* implementation is a completely standard A* with Euclidean distance for the heuristic function.

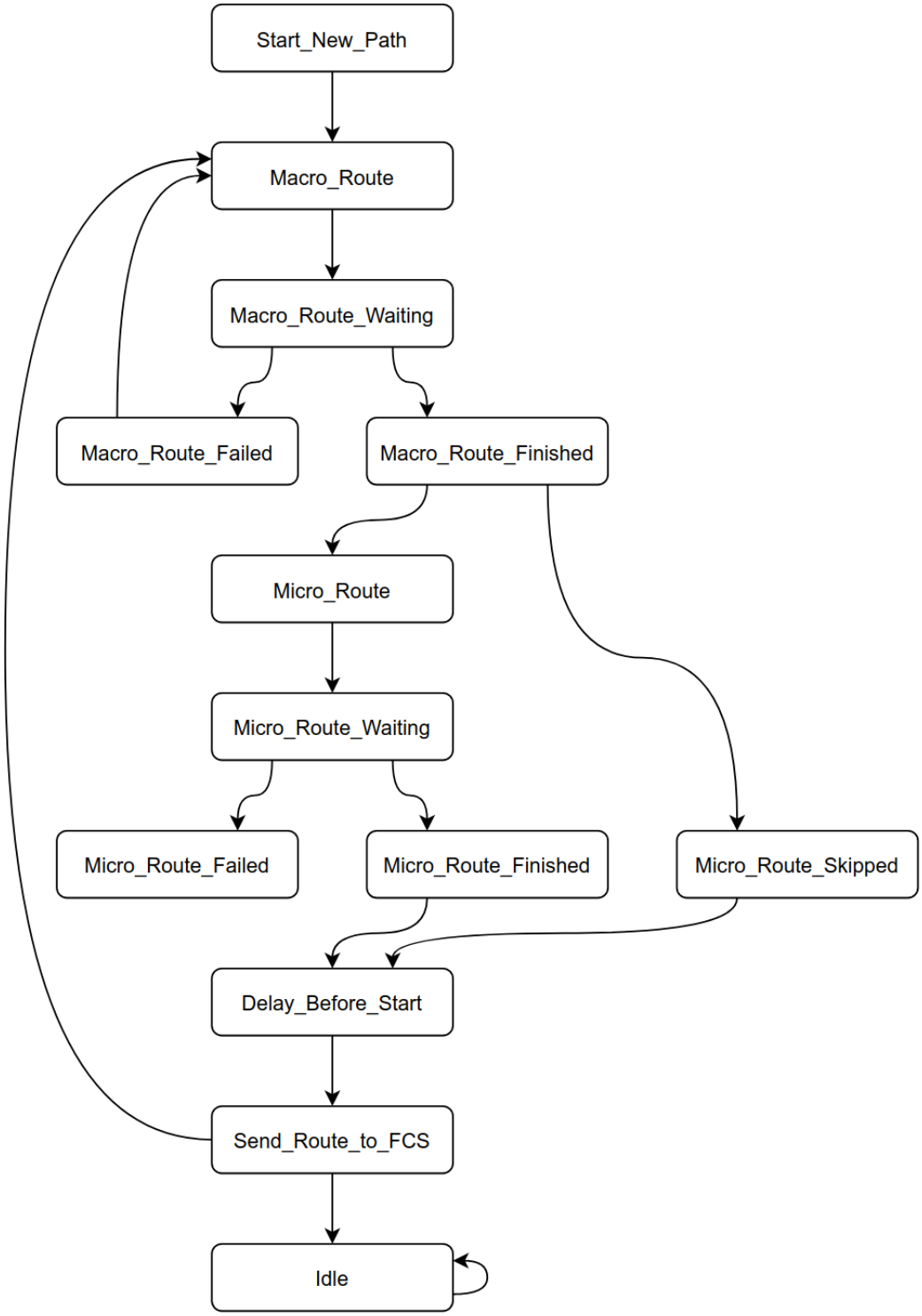

FIGURE 3.19: State Transition Diagram of the Commander's Architecture and Architecture + Scaling modes of operation

Chapter 4

Testing & Analysis

This chapter documents the collection of tests done upon system components to validate their performance, measure error levels for use in state estimation, and identify other notable characteristics. It is organized approximately by order of operation in the system: inputs first, internal processes second, and outputs / full-system testing last. The aircraft is treated as an input for this purpose, since it is a source of motion information and imagery.

4.1 AprilTag Testing

The ability to detect AprilTag markers, and to estimate the markers' position and heading, is critical in this localization system. The case for this capability, and the design submitted to fulfill this requirement, is documented in section 3.2. What follows is a description of testing done to analyze the quality and limits of position data acquired via the prototype AprilTag system.

Certain factors limit the aircraft's ability to detect markers: decreasing distance, large marker size, minimal off-axis angle of the aircraft against the marker, and large resolution of the aircraft's image sensor. Formula 3.7 developed in Section 3.2.3 is believed to have predictive power with regards to the ultimate limit distance for successful AprilTag detection, given all other relevant variables. This formula is restated here for convenience:

$$\triangle X_{h,n} \approx \frac{1}{57} \cdot d \cdot \frac{F_h}{(r_h/\omega)} \cdot cos(a_h)$$
$$\triangle X_{v,n} \approx \frac{1}{57} \cdot d \cdot \frac{F_v}{(r_v/\omega)} \cdot cos(a_v)$$

This section describes the experiments conducted to evaluate this hypothesis. Appendix A covers additional preliminary experiments of a similar nature, but aimed at identifying the specific effects of varying individual parameters.

4.1.1 Test Descriptions and Results

First, two preliminary experiments are done based on the hypothesis that *AprilTag detectability in imagery can be boiled down to having enough pixels*. In this test, an array of several hundred AprilTags is photographed, and then resized to identify the smallest scale at which AprilTags can be decoded consistently. This scale is stated as a "Pixel-per-Pixel" ratio, or the of linear pixels occupied by a single AprilTag pixel. Fig. 4.1 shows the image used as a basis for this test.

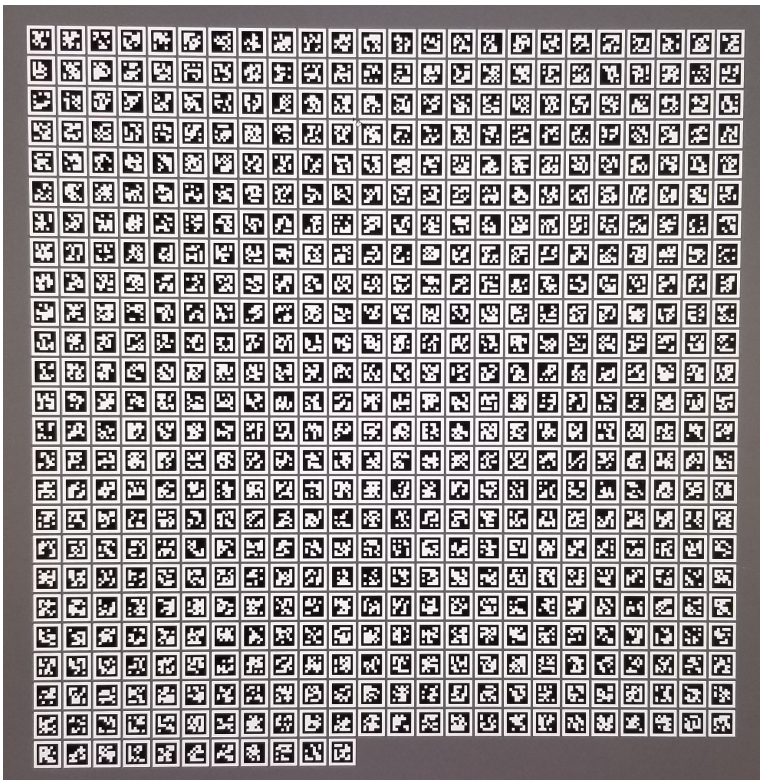

FIGURE 4.1: The original test image used in the first AprilTag pixel-ratio experiment.

Data collected from this experiment is presented in Table 4.1. One can conclude from this table that a Pixel-per-Pixel ratio of about 1.5 is sufficient to reliably detect AprilTags in photos, since the measured values display a distinct curve that approaches 100% at that Pixel-per-Pixel ratio, and remains there for all larger values.

AprilTag Pixel Length in Image Pixels	**Detection Rate (%)**
0.88	0
1.06	10.9
1.25	72.4
1.38	98.9
1.5	100
1.88	100

TABLE 4.1: Images per per pixel at maximum working distance under various parameter combinations

Second, testing was done to evaluate Formula 3.7 in Section 3.2.3 as a predictor

of maximum detection distance. This experiment is also set against the same hypothesis (*AprilTag detectability in imagery can be boiled down to having enough pixels*). It is presumed that all of the accumulated effects of all the discussed variables can still be reduced to their ultimate effect on AprilTag pixel size in captured images; and that that pixel size still predicts decoded success via the same curve documented in Table 3.1.

In each iteration, the camera resolution, field-of-view, target size, and target angle were set. Subsequently, the distance from the target was varied to final the terminal point where AprilTag detections become unreliable. This distance, along with the various inputs and the calculated pixel-per-pixel ratio, are recorded in Table 4.2. Note that for these tests the vehicle was not in flight, but was on the motion-controlled sled in order to enable precise position measurement without using the state estimation system under construction.

Code Pixel Size (mm)	**Resolution (horizontal pixels)**	**Field-of-View (deg)**	**Angle (deg)**	**Max Working Distance (mm)**	**Image Pixels per Code Pixel**
7.5	1280	85	0	3820	**1.69**
7.5	1280	85	15	3580	**1.75**
7.5	1280	85	30	3270	**1.71**
7.5	1280	85	45	2660	**1.72**
5.0	1280	85	0	2390	**1.81**
5.0	1280	85	15	2390	**1.74**
5.0	1280	85	30	2010	**1.86**
5.0	1280	85	45	1610	**1.89**
7.5	640	87	0	1900	**1.66**
7.5	640	87	15	1690	**1.81**
7.5	640	87	30	1550	**1.77**
7.5	640	87	15	1200	**1.86**

TABLE 4.2: Images per per pixel at maximum working distance under various parameter combinations

This experiment suggests that there is predictive power in the equation being tested, as indicated by the relatively constant values for "Image Pixels per Code

Pixel". This consistancy indicates that the formula's predicted limit distance and the measured limit distance agree with each other consistantly, with a scale factor of 1.7 to 1.8.

This actual number - 1.7 to 1.8 - is a slight discrepancy from the figure of 1.5 obtained from initial Pixel-per-Pixel experiment documented in Table 4.1.

4.2 Visual Odometry Testing

This section describes testing done to quantify the accuracy and precision of the ZED Stereo Camera's velocity data. Note that as a third-party component, the ZED Stereo Camera is not fully described within this document, however it is fully described by the manufacturer [105]. Visual Odometry data generated by the ZED camera is consumed by the State Estimation system documented in Section 3.4, and its depth-mapped imagery is consumed by the Occupancy Grid system described in Section 3.5 and the Anti-Collision system described in Section 3.6.

The Visual Odometry System, which is part of the ZED Camera, outputs X, Y, and Z velocities and rotations in the camera's axis at 30 samples per second. This data is generated using propriety algorithms, but is assumed to be based on principles of Visual Odometry - identifying and tracking features between frames, and then computing the camera movement that best matches the features' changes of position. For this research, the ZED system is configured with its world-mapping features and internal IMU disabled.

4.2.1 Noise and Accuracy

This series of tests relies on generating known motions for the ZED to measure, and then analyzing the measurements taken to see how closely they correspond to the known truth, and to analyze their noise levels. To create repeatable and highly-precision known motions, a robotic sled was created. By mounting the sled on a flat and linear track, motion is constrained to a single dimension. Additionally, by driving the sled using a stepper motor and timing belt, exact velocities and accelerations can be created. The sled can be configured for various maximum velocities and accelerations, and can be commanded to move in either direction, or to move to a specified coordinate. This rig is shown in Figure 4.2.

FIGURE 4.2: The "sled" created for use as a known motion source.

Test motions performed were:

- Constant forward motion
- Constant lateral motion
- Constant yaw rotation

The forward motion test determined that when given an actual velocity of -0.5 ft/sec, the measured velocity was within 0.4% of that value. The variance, however, was 17.5%. The data for this result is visualized in Figure 4.3. The perpendicular motion test yielded similar results, with a velocity error of 5.9% and a variance of 12.9%. These results show that although the visual odometry system measures the correct mean values, its moment-to-moment variance is high. Without further processing, it would introduce noise into the flight control system that would be visible as random, jerky aircraft movement. This needs helps to motivate the State Estimation system used in the final system and described in Section 3.4.

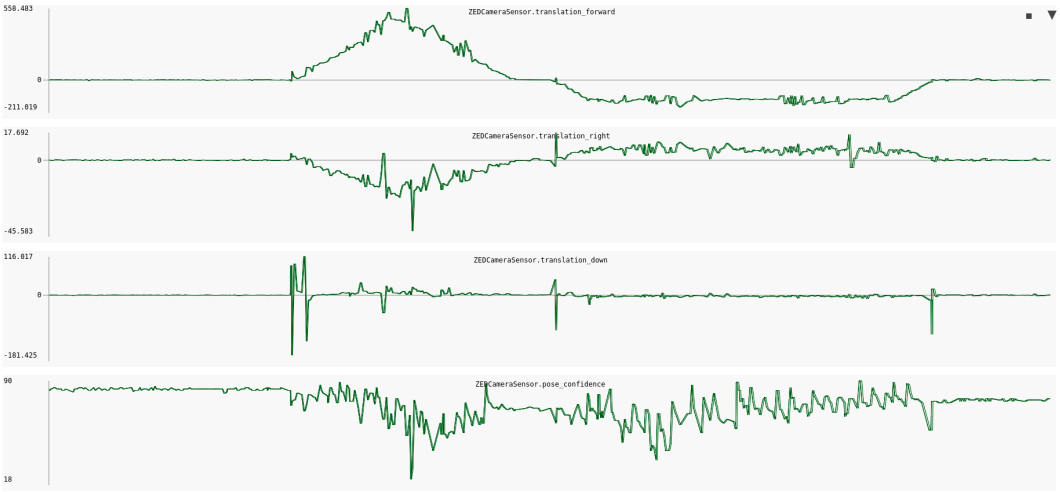

FIGURE 4.3: ZED Measurements from aboard the sled, as it goes through constant-acceleration and constant-velocity states, with the movement axis being aligned with the camera. Note the significant noise and glitches present in all signals. Linear data units are millimeters.

When looking at rotation, a perfectly constant input wasn't available, but this was approximated through repeated trials and then selection of the most constant IMU-measured rotation rate. Several of these trials are visualized in Figure 4.4. One period of constant rotation was analyzed, and found to have a variance of 0.32 deg/sec. Note that these measurements display some data glitches beyond normal error bounds which should be detected and removed.

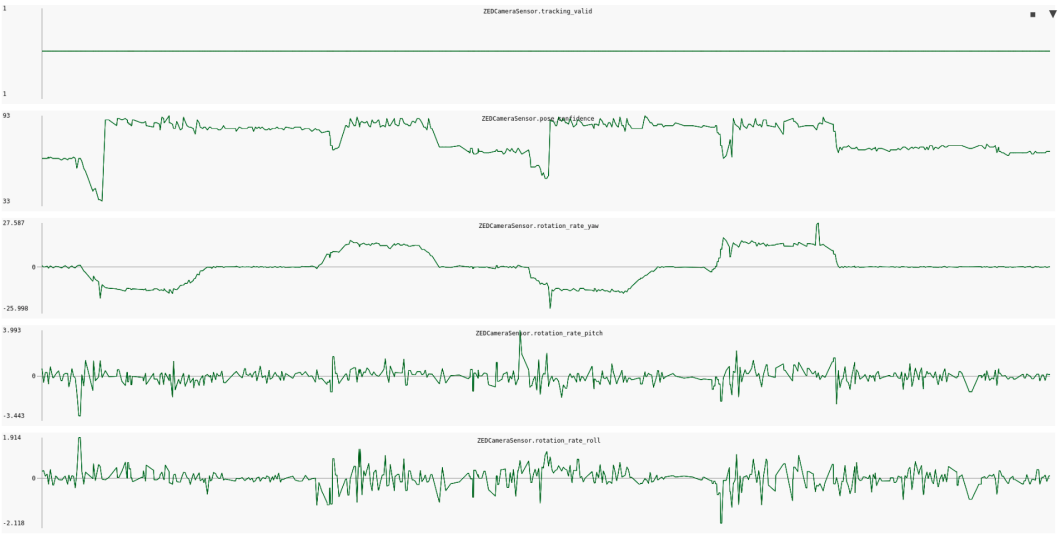

FIGURE 4.4: ZED Measurements from aboard the sled, as it goes through approximately constant-rotation movements on the yaw axis. The middle trace shows the measured rotation rate on the axis of rotation, while the lower two traces show the measured rotation rates on the two other axis, which should be zero. Rotation rates are in degrees per second.

ZED Reset Phenomenon

A significant discovery from this testing was the existence of sustained drop-outs. The root cause of these is not known, but they seem be associated with sudden movements of the ZED camera. They last for approximately one second, after which normal motion tracking is resumed. They are detectable through the "tracking_valid" variable output by the ZED system, which becomes false at the same frame as the data drop-out begins. Figure 4.5 shows an example of this behavior. Here, the ZED measurements show two sustained tracking failures, as indicated by the tracking_valid trace going to zero. These failures are approximately one second in length.

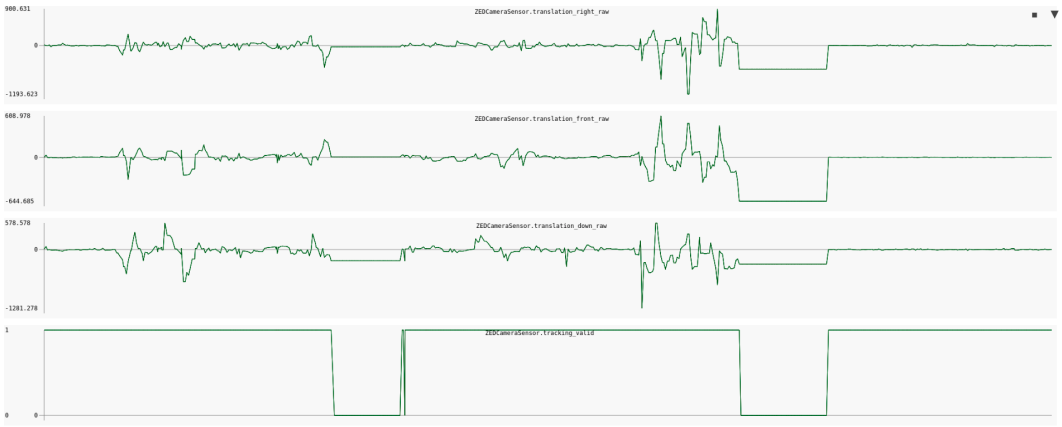

FIGURE 4.5: ZED Measurements from aboard the sled, with induced tracking failures. Linear data units are millimeters.

Identifying these data dropouts and removing them, such that the ZED driver emits no data during these times rather than incorrect data, is essential to properly communicate the available information to the state estimator.

4.3 Delay Solver Testing

The state estimator takes signals from multiple sensors, and fuses them into an estimate of the true state of the overall system. But for that to work, data must be synchronized. The design of the "Delay Solver" created to address this need is documented in Section 3.4.2. The Delay Solver computes the cross-correlation between the first derivative of both signals, and looks for a sharp peak value. Additionally, it has structures to determine if the signals are suitable for analysis, or are too bland to aid in a solution (Note that if the aircraft is sitting still and both traces are straight lines, there is no information available to determine a relative delay).

This section describes the testing done to determine when signals are usable, and testing done to determine the accuracy of the delay solver.

4.3.1 Testing for Correct Operation

To determine the accuracy of the delay solver, a test is performed wherein the vehicle is held still while occasionally being given short and sharp movements, such that those movements can easily be identified in acceleration graphs. Data is captured of the two raw input data streams (IMU and ZED VO), and of the Delay Solvers output, and finally the Delay Solver's internal cross-correlation graph. This data is manually analyzed for correctness.

Results indicate that although the system is technically functioning correctly, as indicated by tracing individual frames of data through the system, the performance over time varies widely. Fig. 4.7 shows typical data seen by the delay solver system. The first data stream in blue is the the ZED Velocity data, converted to acceleration by differentiation, and then reduced from 3D to a magnitude. The second stream in red is the Aries Accelerometer data, also reduced to a magnitude. Applying a cross-correlation with a range of temporal offsets results in the curve in Fig. 4.7, where lower numbers indicate higher agreement between data streams. The minimum value on this curve identifies the best time offset to fit the data.

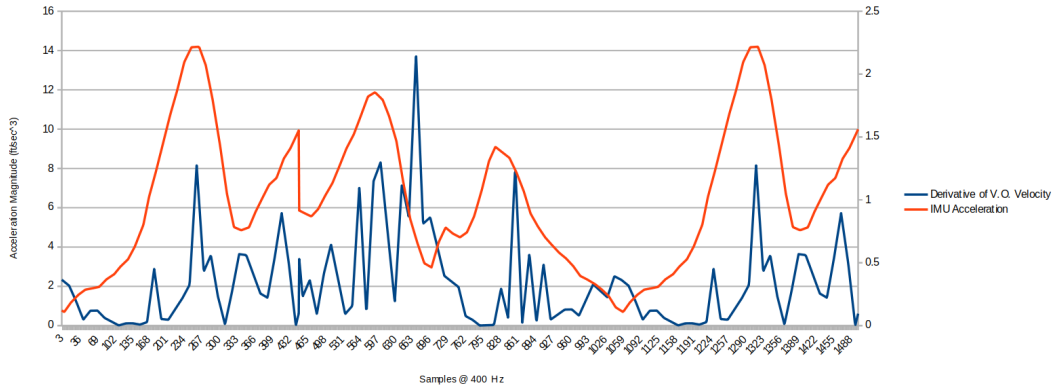

FIGURE 4.6: Sample pair of data streams after pre-processing by the Delay Solver.

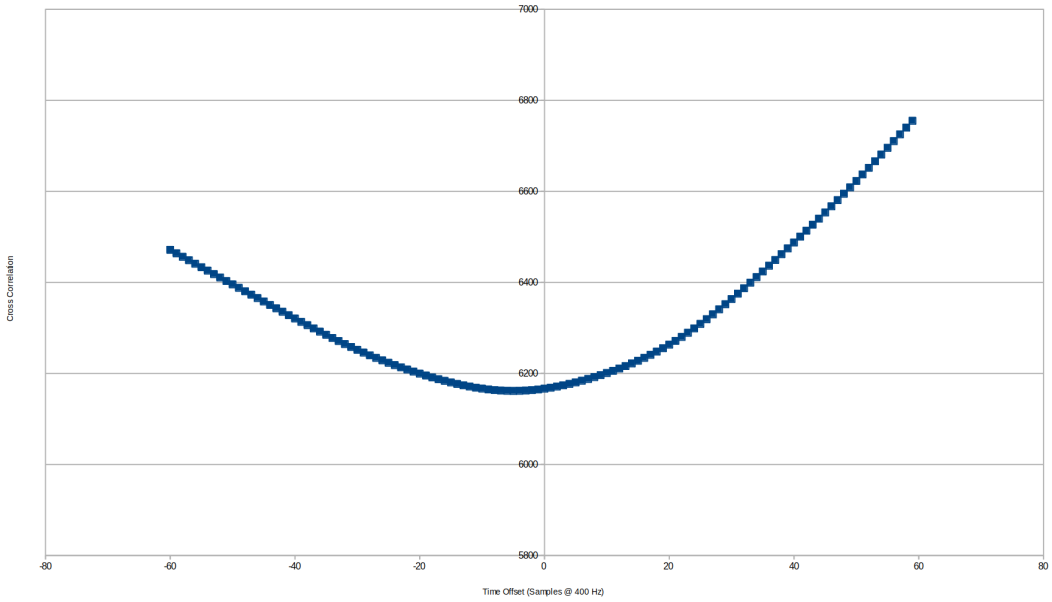

FIGURE 4.7: Cross-correlation of the two data streams at differing time offsets. The X-axis shows the time offset of the second stream relative to the first. The value plotted on the Y-axis is the sum of the differences between sets of matching samples. Lower values indicate that the two streams are more similar when the second stream is given the specified time offset.

Unfortunately, if the system is observed in operation over time, the results do not converge on a specific time offset as expected, but move randomly through the entire range of possible values. Further research is needed to hunt for algorithmic problems in the Delay Solver or in related systems. Further research could also be performed to examine the possibility that the variable delay is correct, indicating that other components of the system have timing or reliability problems. Fig. 4.8 illustrates this with in-flight data.

FIGURE 4.8: Delay Solver "decisions" over time, showing a lack of agreement on any single value. Values are in milliseconds.

4.4 Aircraft Testing

4.4.1 Desired Information

This section describes the performance of the test aircraft, insofar as it is relevant to the effectiveness of the Visual Navigation system. Although the Aircraft is conceptually at output side of the VNS, its Inertial Measurement Unit (IMU) is also used as a data source, and thus must be understood. The UAV used for this evaluation is composed of a DJI Flamewheel Multirotor frame and power system, with the Aries, Jetson, and ZED hardware described in this paper in Section 3.1.

It is hypothesized that the aircraft's vibration, level of angular control, and visual obstructions could affect the Visual Navigation system.

4.4.2 Vibration

Vibration can be defined as movement beyond the frequencies that the drone's control loops are designed to operate in. Movement, weather angular or linear, is controlled from 0 to approximately 25 hz, and the flight controller applies a lowpass filter to incoming movement data in an attempt to exclude vibration. Nevertheless, vibration leaks through the filter and degrades the aircraft's stability, which in turn affects anything mounted to the aircraft, such as the Visual Navigation camera(s). More directly, vibration may be transmitted into those cameras, resulting in blurred images. To measure vibration, a high-pass filter can be applied to incoming movement data, and then the RMS average signal level over time can be computed.

Tests are performed with the aircraft in flight while holding position, heading, and altitude, such that the noise produced by it's motors and propellers is most representative of what is typically experienced in flight. Separate measurements are taken for each of the three rotational axis and three linear axis. Two sets of flights are performed: one with the aircraft in ground effect, six inches above the ground, and one with it clear of ground effect, six feet above the ground. Prior to testing, propellers have been balanced to minimize individual rotor vibration as much as possible. The IMU is sampled at 1000 Hz.

The following tables 4.3 and 4.4 present the vibration levels measured during this experiment:

Sensor	Axis	RMS Vibration Magnitude
Gyroscope	Roll	6.1 deg/sec
Gyroscope	Pitch	10.2 deg/sec
Gyroscope	Yaw	5.9 deg/sec
Accelerometer	Right	0.18 G
Accelerometer	Front	0.26 G
Accelerometer	Down	0.48 G

TABLE 4.3: Measured vibration levels from aircraft IMU. Aircraft hovering in ground effect at 6" above floor

Sensor	Axis	RMS Vibration Magnitude
Gyroscope	Roll	6.5 deg/sec
Gyroscope	Pitch	11.3 deg/sec
Gyroscope	Yaw	5.7 deg/sec
Accelerometer	Right	0.16 G
Accelerometer	Front	0.22 G
Accelerometer	Down	0.47 G

TABLE 4.4: Measured vibration levels from aircraft IMU. Aircraft hovering in clean air at 6′ above floor. Based on this data and the data in Table 4.3, standard values were determined, to be used as expected variance on all IMU samples.

4.4.3 Angular Control

Data was gathered on the angular stability of the test aircraft, but this data was ultimately unneeded in the final system design. It has been preserved as Appendix B.

4.4.4 Visual Obstructions

This experiment identifies permanent visual obstructions expected in the ZED camera during normal operation. If the drone itself, rather than the surrounding environment, fills too much of the camera's image, Visual Odometry can be affected. Additionally, the drone hides useful features of the environment including AprilTags whereever it is visible.

The images in Fig. 4.9 and 4.10 show a representative view of the aircraft's camera, and a generated mask separating pixels obscured by the aircraft from pixels which can see the environment.

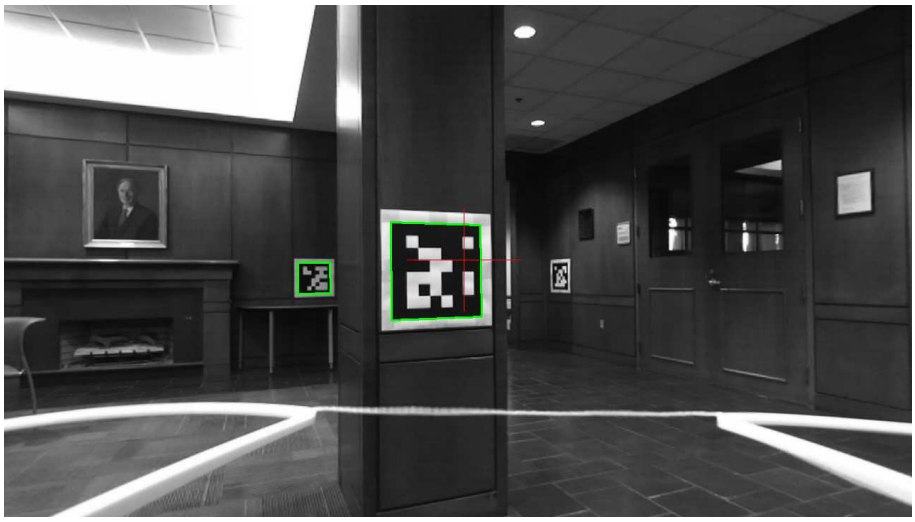

FIGURE 4.9: Typical image captured by the aircraft's ZED camera

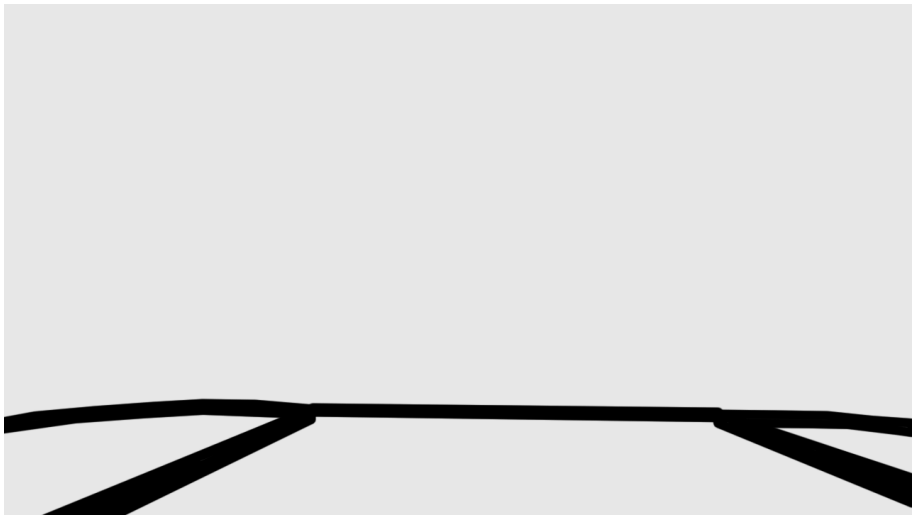

FIGURE 4.10: Segmentation of the ZED camera image, where black pixels are pixels which view the aircraft's body rather than the surrounding environment, and thus are unusable for the intended sensing operations. Note that under indoor lighting conditions, the propellers are not visible in flight and do not obscure the camera.

4.5 State Estimator Testing

The state estimator combines information from multiple sensors, as described in Section 3.4. The individual sensors where characterized, particularly with respect to signal variance, in the preceding sections of this chapter. Each sensor measures some aspect of the aircraft's state, while being subject to noise and latency. The state estimator attempts to compute the true state of the aircraft from these measurements.

Preliminary testing focuses on steady-state performance: is the output accurate? How do noise levels compare to the raw signals? Are any adverse conditions such as smoothing or latency observed?

Subsequent testing focuses on edge conditions. The estimator's behavior can be characterized during a partial loss-of-data event. The state estimator can also be examined in the presence of disagreeing data, such as two AprilTag position markers, that indicate substantially different positions.

Note that section 4.10 contains complimentary material. It covers the flight abilities of the aircraft while experiencing temporary sensor failures, which is reflective of the performance of the state estimator under those conditions.

Accuracy

To check for correct functionality, constant-velocity linear movements are performed using the linear motion sled pictured in Figure 4.2. The average velocity, as measured by the ZED motion tracking system, is compared to the average velocity from the state estimator. Next, three-phase motions are performed. The phases are: constant acceleration, constant velocity, constant negative acceleration. Jerk is high enough to have negligible effects on the actual velocity, when compared to an idealized infinite-jerk model. With this motion pattern, it can be determined if the state estimator is capable of tracking sharp "corners" in velocity. This test can also be used to determine if the state estimator develops any bias during motion, as such a bias would result in the state estimator returning to a non-zero velocity estimate when stopped.

Since the state estimator calculates states as Gaussian distributions, both the mean and the variance should be examined. The variance, in general, should decrease with each new sample of data that arrives, and should increase over time in

the absence of new data. Given a steady stream of data, the variance should find an equilibrium point between these two forces.

Basic correctness of the state estimator was verified through data visualizations such as Fig. 4.11 and 4.12. When given a known input, such as a constant velocity movement or trapezoidal acceleration curve, visual inspection is sufficient to determine the general correctness of the state estimator. Additionally, mean values are checked for correctness, and operation is verified under multiple headings and attitudes, to guard against incorrect vector rotations.

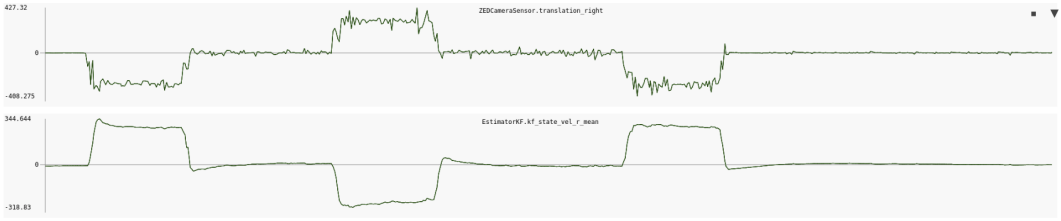

FIGURE 4.11: Periods of constant-velocity movement, showing the raw vs. estimated state. The estimated state has removed much of the noise from the input signal, but displays a slight overshoot behavior after sudden accelerations. The flipped signal polarity is a side-effect of the output being in a difference reference frame from the input.

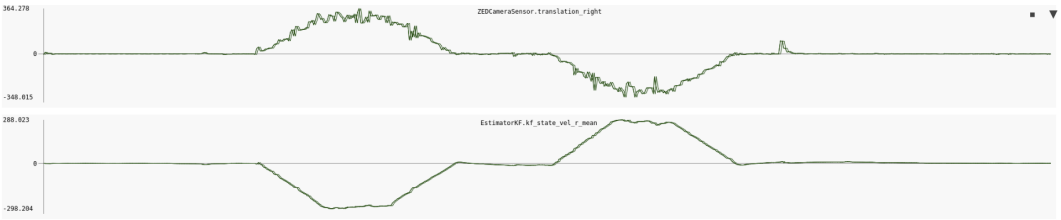

FIGURE 4.12: Periods of trapezoidal-velocity movement, showing the raw vs. estimated state. The estimated state retains low-latency behavior even as it smooths the velocity data. This is contrast to a hypothetical low-pass or moving-average filter, which would induce latency and "round off" sharp corners in the data. The flipped signal polarity is a side-effect of the output being in a difference reference frame from the input.

Much attention was given to the behavior of variance estimates during operation. The variance of individual samples must be known, and this data can be pulled directly from a data such as the ZED camera. For data sources that do no provide an uncertainty estimate, such as the IMU, this value must be measured and stored, as exemplified by the study of in-flight IMU data in 4.4. Fig. 4.13 shows a captured state variance that has reached an equilibrium during normal operation, with the variance decreasing with each new sample but increasing over time. Fig. 4.14 shows

the result of a sensor failure, which causes the related variance to increase without bound, correctly indicating that the system is increasingly uncertain of the correct value for that state variable.

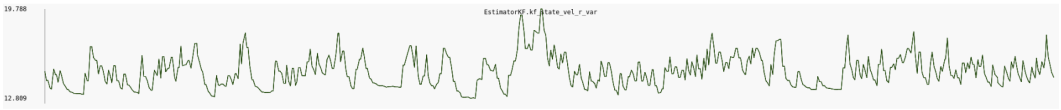

FIGURE 4.13: Variance during steady-state operation. Variance increases over time, but decreases with each new data sample. The result is an essentially constant value, with some expected variation due to timing jitter.

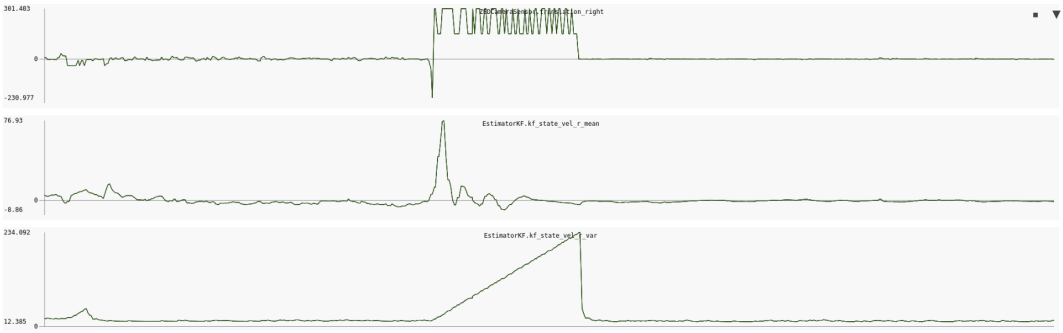

FIGURE 4.14: Variance during failed-sensor operation. Variance increases over time without bounds, indicating that the state estimator is increasingly uncertain about its output. Note that this means the state estimator is still working despite the failed input, since it is correctly predicting a state which has a mean value but also a large and increasing expected error.

4.6 Occupancy Grid Testing

During development of the occupancy grid system described in section 3.5, several hurdles had to be overcome. First it had to be shown that depth-mapped pixels were being accumulated into the correct voxels, given a vehicle's position and attitude, and a pixel's position in the image.

Next, a system of data accumulation had to be developed for each voxel, to determine when enough information was available to make a decision about the voxel's occupied/vacant state, and what that state was. That system should be fast enough for real-time operation, including situations where the physical environment changes during operation. Yet it should be steady enough to not display excessive noise in the voxel field, which would be problematic for path planning.

Finally, decisions needed to be made about the size and resolution of the voxel field with respect to the computational capabilities of the aircraft. As a fully-populated 3D array, changes in size and/or resolution have large impacts on computational requirements. A related system shifts the rectangular prismatic "area of observation" by units of one voxel to follow the drone, while maintaining the state of the voxels within.

Final acceptance testing was done to determine that when the aircraft was allowed to "discover" obstacles, that the rendering of those obstacles in the GUI matched the real-life obstacle.

Note that these tests are done in the spirit of determining if the system is working properly, rather than in the more scientific aspiration of quantifying all aspects of the system. Results are correspondingly summary.

4.6.1 Single-Voxel Behavior

To understand the behavior of the occupancy grid system as it accumulated data, it is useful to focus on a single grid square, and with virtual pencil and paper simulate that square over time. Although an full-system test is ultimately required, the amount of data presented can tend to obscure individual details. This "test" uses the

final application logic to determine the behavior of a grid square under certain typical conditions: distant obstacle, nearby obstacle, nearby clear, out-of-view obstacle, out-of-view clear.

To perform this test, the occupancy grid software is modified to record the mean and variance of a single chosen grid square, along with the it's current distance from the aircraft, if it's in-frame or not, and if it's current a detected obstacle, detected clear, or hidden behind closer obstacles. This data is graphed and presented along with an analysis of the system's behavior over time. Fig. 4.15 and 4.16 show how the system builds confidence in a voxel's occupancy when consistently registering "hits", and that the rate of trust depends on the distance from the voxel, reflecting the system's lower certainty of more distant depth samples. Fig. 4.17 shows that the same action occurs when also actively "missing" the voxel, indicating that the voxel is in clear space.

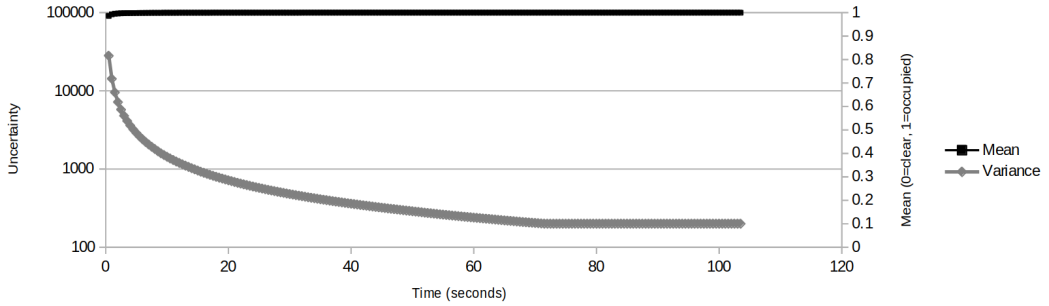

FIGURE 4.15: Experimental Voxel State Measurement Under Constant Depth Hits, Distance = 10 ft. The mean value quickly rises to 1 (meaning that the voxel is occupied) due to the lack of any alternative information. At the same time, confidence slowly increases over time and repeated samples, until ultimately hitting its lower limit.

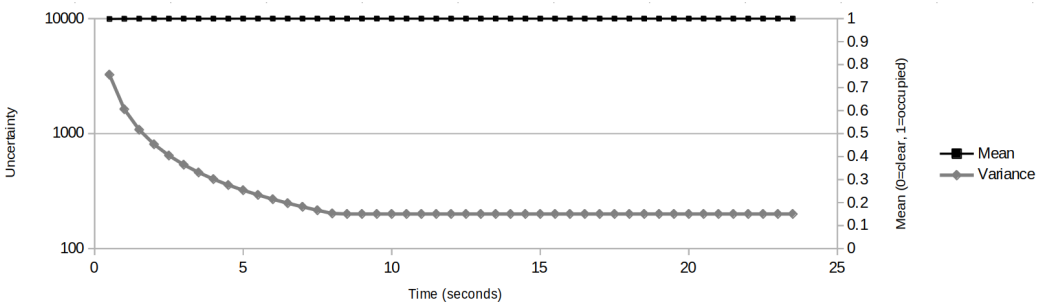

FIGURE 4.16: Experimental Voxel State Measurement Under Constant Depth Hits, Distance = 5 ft. The results are the same as the 10-ft experiment in Fig. 4.15, but the system converges to a result more quickly due to the higher-quality samples afforded by the smaller distance.

Fig. 4.17 shows that the voxel's state remains unknown (ie. its value is halfway between "occupied" and "clear"). The voxels variance does decrease over time for unknown reasons, but at a very slow rate. This phenomem probably reflects a bug that should be corrected.

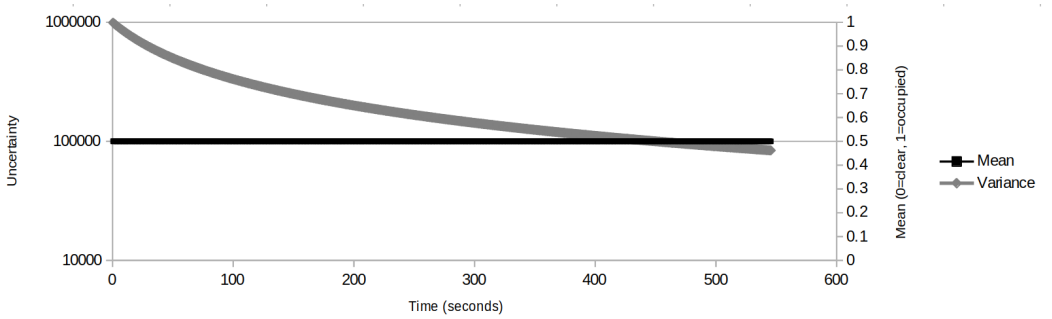

FIGURE 4.17: Experimental Voxel State Measurement Under Constant Depth Misses

The preceeding experiments were all based upon motionless situations. To further validate the system's ability to assign lower uncertainties to nearby depth samples, the drone is moved towards the object at 1 ft/sec as data is collected. The graph (Fig. 4.18) shows that variance accelerates its descent as compared to the state tests above.

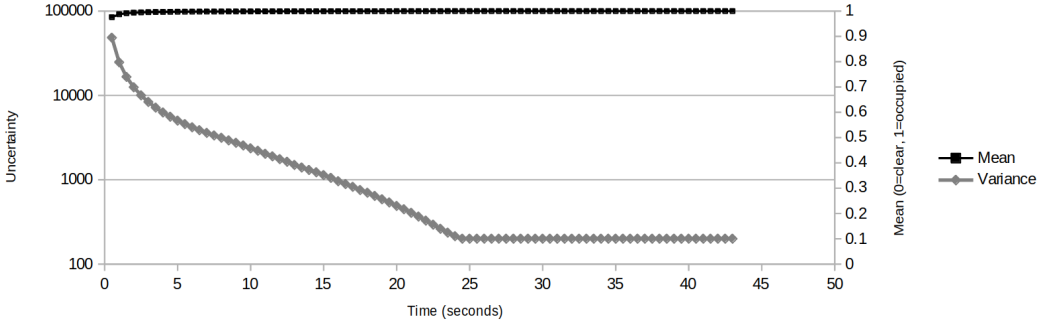

FIGURE 4.18: Experimental Voxel State Measurement Under Constant Depth Hits, Distance = 10 ft Decreasing to 5 ft

The final experiment shows that the voxel remains capable of updating its state, even after initially "settling" on a Clear or Occupied state with low variance. In this test, a person physically moves in and out of the target voxel several times, causing the sensed data to alternate between Occupied and Vacant. The chart in Fig. 4.19 shows that when the current observed state disagrees with the internal state

estimate uncertainty increases, until eventually the internal estimate changes and certainty starts to increase as that estimate is reinforced.

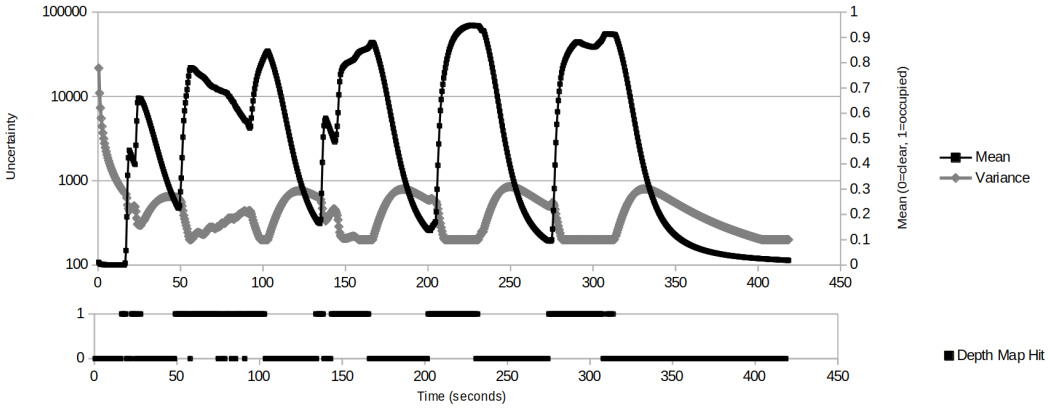

FIGURE 4.19: Experimental Voxel State Measurement While Experimenter Walks In and Out of Target Position. Note the mean value alternating between 0 ("Vacant") and 1 ("Occupied"), while variance rises and falls depending on how much the inputs (shown underneath the main chart) agree with the current state estimate.

4.6.2 Start-up Behavior

Fig. 4.20 shows a typical example of start-up behavior of the system. These six frames of video show the system discovering more blocked voxels over time. Concurrently, the pathfinding systems are starting to develop waypoints to circumnavigate the blocked voxels. This series of images covers a time space of 6 seconds.

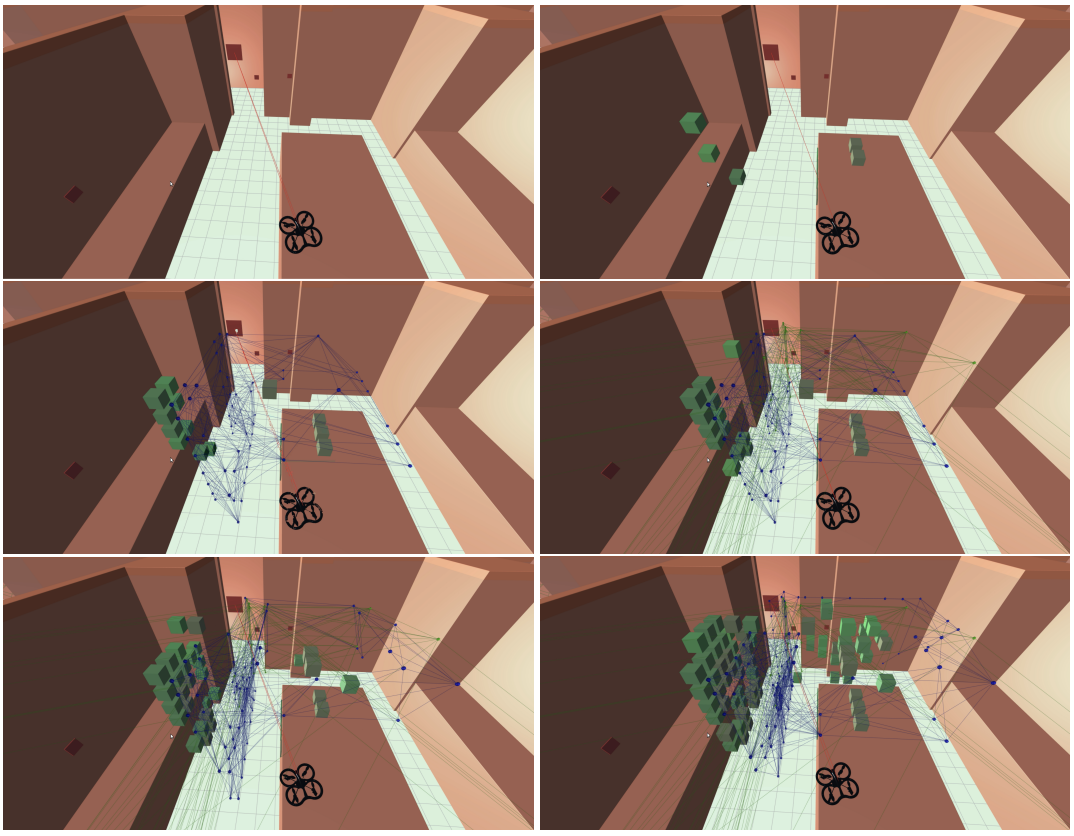

FIGURE 4.20: Start-up Behavior of Occupancy Grid

4.6.3 Computing Resources

The speed, memory, and communication (for remote visualization) requirements of the Occupancy Grid system are dependant on the total number of voxels contained within the "Local Navigation Area". Recall that the Local Navigation Area is a shifting cuboid, with the vehicle at its center. As implemented, the occupancy grid currently is defined upon these parameters (Table 4.5):

Parameter	**Value**
Voxel Size	1x1x1 ft
Grid Length	40 Voxels
Grid Width	40 Voxels
Grid Height	10 Voxels

TABLE 4.5: AprilTag detection success and accuracy versus camera resolution. Aircraft sitting on table, not running.

One can calculate from this data that there are 16,000 voxels in memory. It is believed that all Occupancy Grid operations are of $O(N)$ complexity, where N is the number of voxels. The current system bottleneck if the communication between the ground station and the aircraft for visualizing the voxel field, and the performance of the 3D rendering system on the ground station when many voxels are visible. Without any visualization features, the voxel field should be able to scale to a higher resolution or size without overloading the aircraft-bourne hardware.

The Occupancy Grid's performance is also affected by the size of the depth images, since each pixel of the image (or a fraction thereof if the image is downsized) must be raytraced into the voxel field and can trigger a change of state within the target voxel. (Table 4.6)

Parameter	**Value**
Source Width	1280px
Source Height	720px
Downsampling Ratio	16
Final Width	80px
Final Height	45px

TABLE 4.6: AprilTag detection success and accuracy versus camera resolution. Aircraft sitting on table, not running.

As implemented the Occupancy Grid system runs at an average speed of 0.62 seconds per update cycle, giving it a frame rate of slightly less than 2 frames per second.

4.7 Anti-Collision Testing

The anti-collision system is a low-complexity system designed to prevent detectable collisions without requiring any stateful information or complex logic. Its design is description in section 3.6. The system uses depth sensing and thresholding to determine in an object is dangerously close to the aircraft. If so, the aircraft is not allowed to move forward (but can still move in sideways and backwards)

4.7.1 Visualization

To validate that the system correctly identifies nearby objects and also correctly both distant objects and areas where no depth information is available, images are generated through special programming of the Anti-Collision system. The system has two alert levels: "warning", where a notification is generated without affecting vehicle navigation, and "critical" where the vehicle ceases forward motion and cannot be directed forward. Objects 2-4 feet from the camera are labelled as "warning", while objects 0-2 feet from the camera are "critical". Diagnostic images are created, based on a synthetic depth image, with "warning" and "critical" detections color in yellow and red, respectively. One such image is displayed as Fig. 4.21. Images consistently demonstrate that the anti-collision system is correctly labelling objects.

FIGURE 4.21: Enhanced image of depth sensing, with areas detected as caution (yellow) and critical (red) hazards colored.

4.7.2 Resource Usage

As an image processing task, the Anti-Collision system uses valuable processing resources. In order to limit the impact of this system, the depth image is first down-sampled, and the subsequent thresholding and binning tasks are done against the 1/16th resolution image. Under this configuration, each depth image can be processed by the Anti-Collision system in 2 milliseconds, making its impact on compute resources neglegible.

4.7.3 Small Objects

After testing the anti-collision system successfully against large objects (for example by trying to drive it into a wall), the question was raised of how it would handle less glaring obstacles. To quantify the systems ability to detect small objects, two objects were selected, and the maximum distance at which they consistently triggered an alert of the anti-collision system was found through experimentation. The objects were vertical columns / tubes, one with a diameter of four inches, and one with a diameter of one inch. The results, shown in table 4.7 demontrate sound performance, with the larger object being detectable at all ranges, and the 1" object being detectable at a range of 1.5 feet from the camera (which equates to a range of 9" from the drone's outer edge).

Obstacle	**Maximum Distance (ft)**
4" Vertical Rod	4*
1" Vertical Rod	1.5

TABLE 4.7: Anti-Collision System reacting to small objects. A 1-inch diameter rod triggers the detection system from 1.5 feet away. Note that this distance is relative to the ZED Camera, not the aircraft extents. On the test aircraft, this results in a clearance of approximately 9 inches. *Note that 4 feet is the maximum range of the Anti-Collision system.

4.8 Path Generation Testing

In order to be deemed admissible, the path generation component must:

- Generate a path from the starting point to the destination
- Avoid all given obstacles by a specified margin

The second point means that not only to the vertexes of the generated path be in unobstructed space, but all points along the edges between vertexes must be in free space as well.

This series of tests uses both specially-constructed spaces and real-world spaces to determine if the path generation components meet these criteria for admissibility. It should be noted that these tests are testing the combined system comprised of the actual pathfinding component (the A* implementation), as well as the obstacle generation / expansion / simplification components, since these obstacles are inputs to the pathfinder.

Real-world and specifically-crafted testing environments were selected/created to exercise the path generation system. The output of the system, which is a undirected graph of three-dimensional nodes, is visualized on a 3D GUI. Since obstacles are also visualized on the GUI, visual inspection can be used to determine if any nodes or edges intersect with obstacles. This testing was done during the development process, so new test cases were developed as modes of failure were discovered.

The path generation system's design is documented in section 3.3.

4.8.1 Paths Through VCU Area

Many tests were done with a map based upon a portion of VCU's Engineering West Hall, albeit with many modifications designed to create more unique and challenging navigation. Fig. 4.22 is an example of this, with the pathfinder successfully find a route towards the terminating point at the bottom of the screen. Routes also contain altitude for each point, though that is not visible from this 2D perspective.

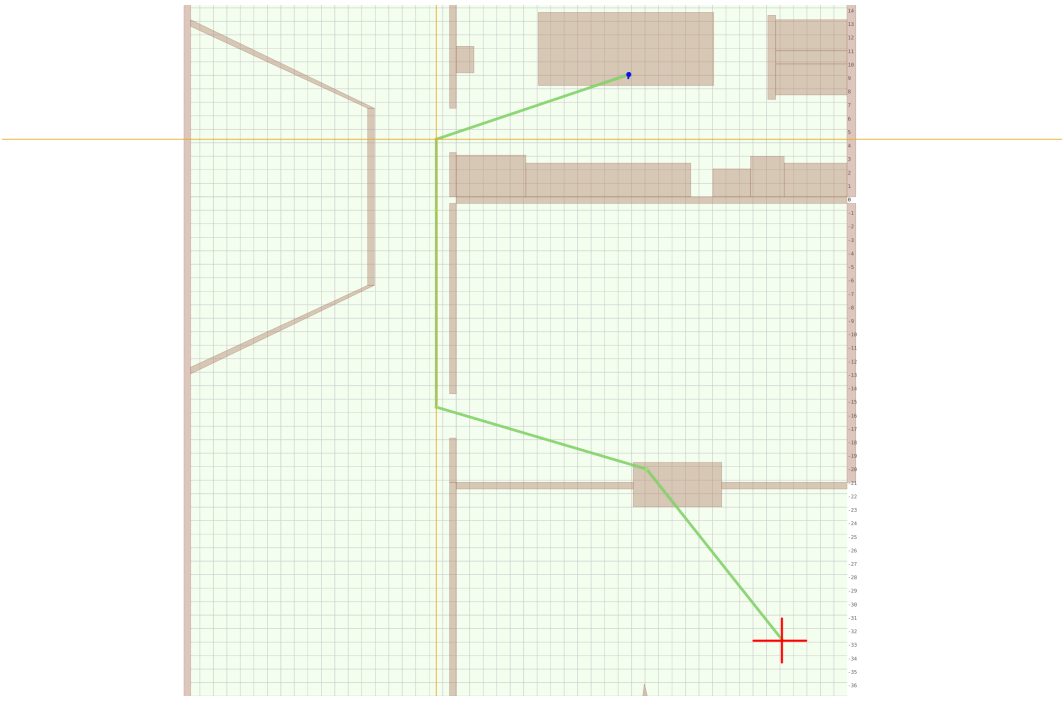

FIGURE 4.22: Successful path generation utilizing open doorways and corridors. Note that the obstructed doorway near the bottom of the screen is not full-height, and so the drone can safely pass through it at specific altitudes.

4.8.2 Paths Requiring Altitude Changes

Fig. 4.23 shows a path where altitude changes are necessary - the path starts at a high altitude, but encounters a doorway that can only be passed through at a low level.

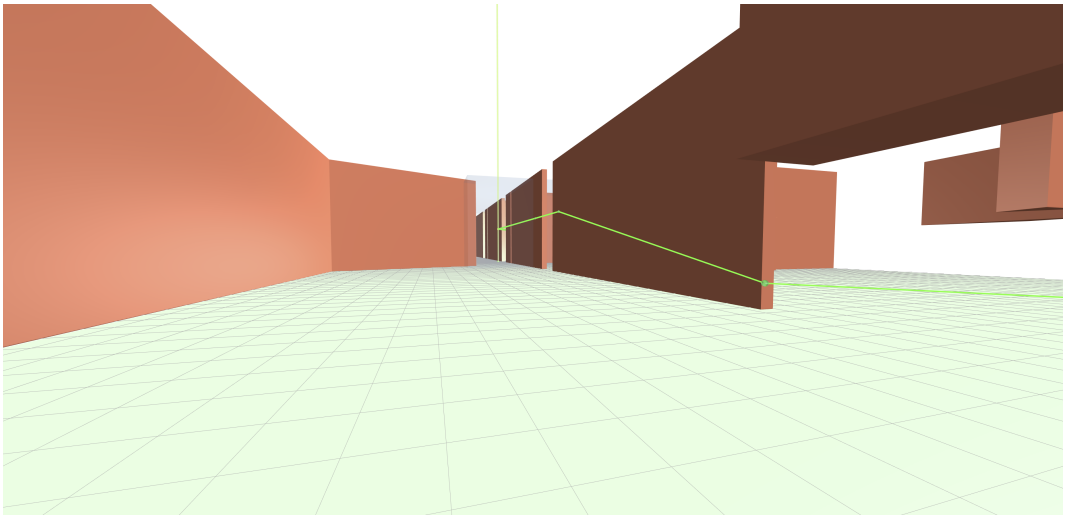

FIGURE 4.23: Successful path generation requiring specific altitudes to traverse specific areas of the map. The doorway on the right edge of the image is designed to force the system to distinguish high-altitude, inpassible edges, from low-altitude, passable ones.

4.8.3 Impossible Paths Due to Complete Blockage

In addition to these successful route generation tests, it is critical that the system correctly identify situations where a route cannot be found, and refrain from output partial paths or false directions. Fig. 4.24 illustrates this situation.

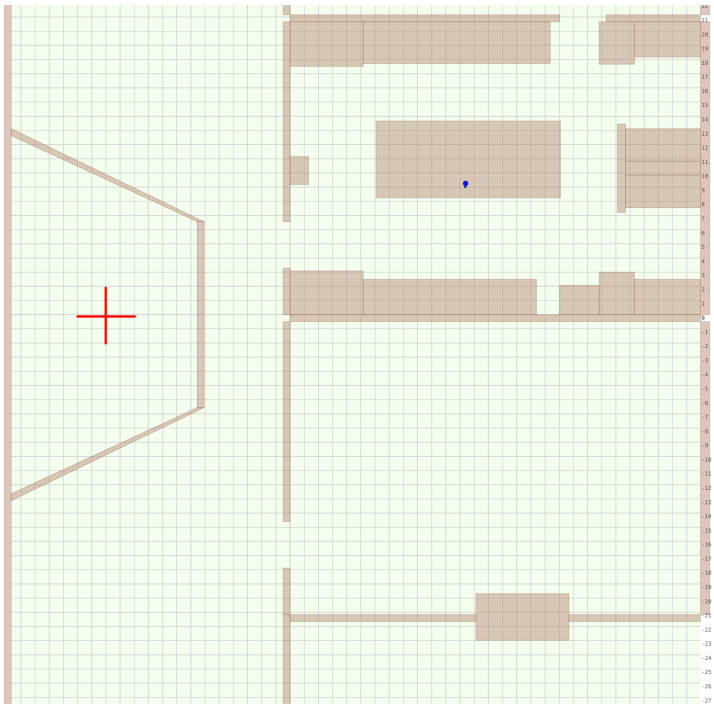

FIGURE 4.24: The target waypoint is set in the middle of the acute trapezoidal room on the left edge of the map. Since there is no possible route from the drone's current position (blue dot) to the target, the system correctly outputs nothing and does not move the drone.

4.8.4 Impossible Paths Due to Insufficient Safety Buffer

Unsuccessful routing attempts can also result from situations where an area may not be totally blocked, but the user-defined safety buffers around all solid objects result in less-than-zero clearance for some section of a potential route. For example, a narrow corridor with wide safety zones around the walls may have no usable space inside. Fig. 4.25 shows two views of the same map, with the "route planning" option enabled where all potential route segments are displayed. The two images have different safety buffer sizes defined, and one can see that in the image with smaller safety buffers, more routes are available.

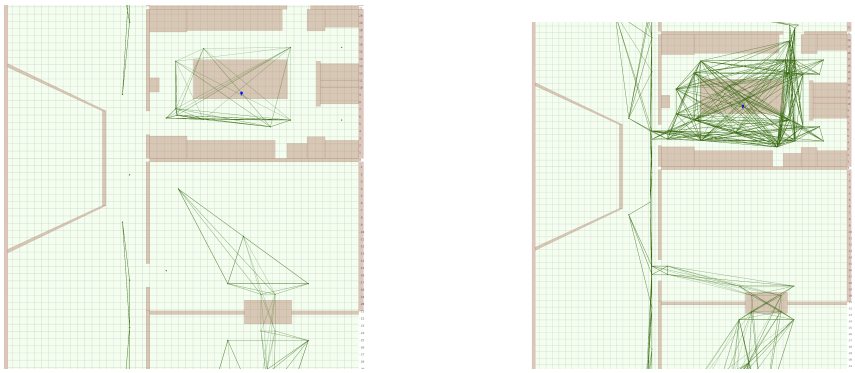

FIGURE 4.25: Two images of route planning on a map. The left image has large safety buffers around solid objects, and thus has fewer options for navigation. One can also see the size of the safety buffers as clear space around the walls and solid objects.

4.8.5 Two-Stage Planning Avoiding Local Obstacles

There were two phases of testing. Phase one tests used a single pathfinding instance, and were intended to test the correctness of the pathfinder and its interactions with the environment. Phase two tests used the final two-stage path generation described in this paper, where a large-scale path is determined first, and then a small-scale path generation is done to determine a path through a high-resolution grid of nearby obstacles to the current large-scale waypoint. One must remember when viewing these results that the large-scale path is allowed to cross through voxel obstacles, as long as the small-scale path is ultimately be able to find a way around the voxels.

To test the two-stage system, a plan was developed to construct a scenario where there exists two routes between the source point and the target. One route is short and straightforward, and another route is much longer and more costly. After confirming that the system initially picks the shorter route, which is correct behavior, a door is closed (physically) and the system is allowed to discovered through its sensing system that the doorway is no longer passable. At this point, the system should re-route using its next-best option, the longer route. Fig. 4.26 shows images of this test.

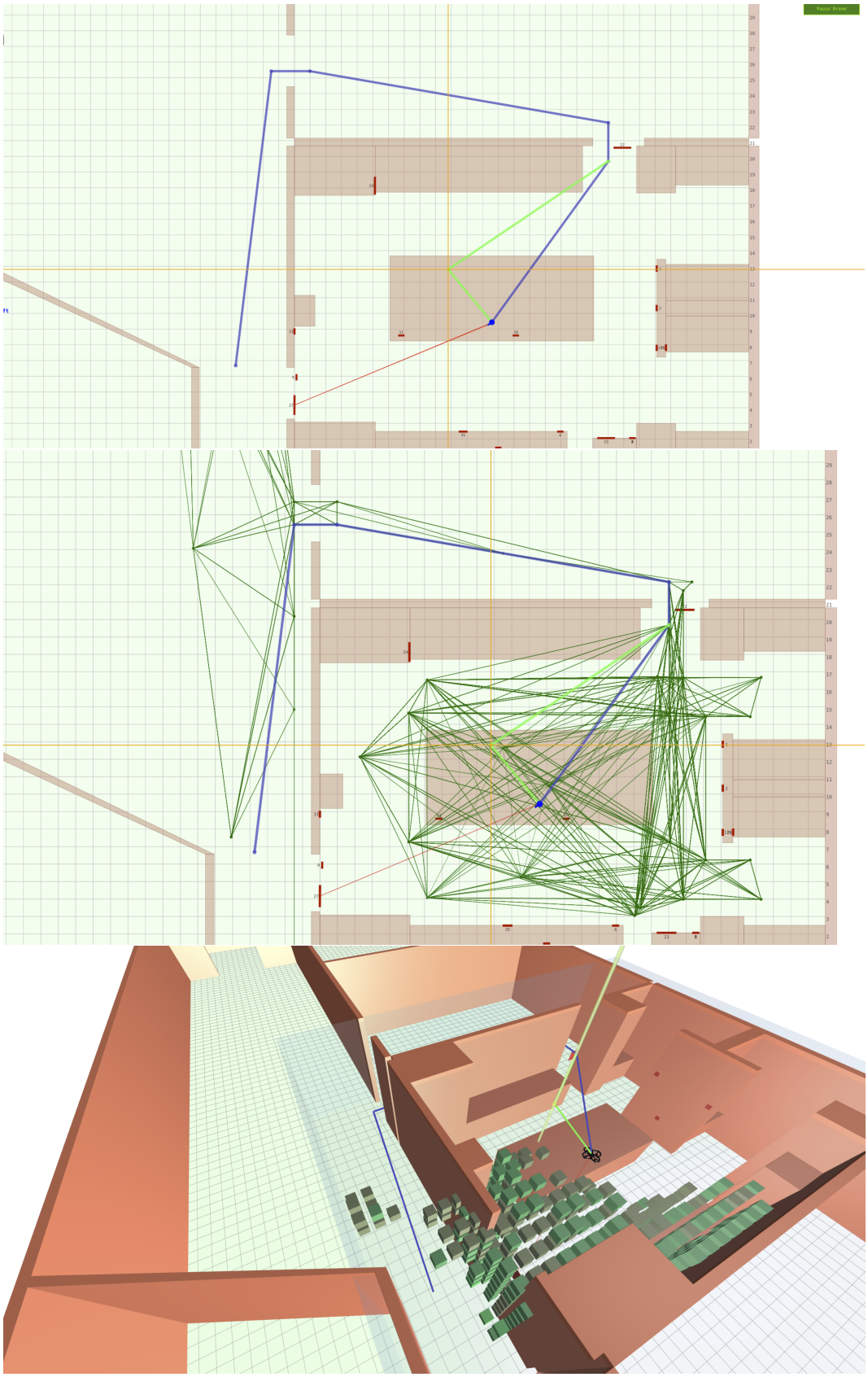

FIGURE 4.26: Long path chosen due to an obstructed doorway. The second image shows the path planning view, which reveals that potential path segments near the obstructed door have been removed from consideration. The final image is a 3D view showing the detected blockage points.

4.9 Full-System Testing

These tests attempt to quantify the performance of the full localization and navigation system under typical workloads. The full system is constructed as per section 3.1.

4.9.1 Position Hold Performance

The aircraft's ability to hold a position is a product of its sensory input, its on-board state estimation, its control algorithms, and the physical capabilities of the aircraft. Previous tests in this report have quantifies many of these individual components, yet the full-system performance is still the ultimate deciding factor when determining the success of the system design. Both the absolute correctness of the aircraft, in the form of minimal position errors, and the smoothness of the aircraft, in the form of minimal velocity errors, are desirable.

Testing the performance of the aircraft in position-hold mode is complicated by the need for a reliable method of measuring the aircraft's actual position. A method was developed where an overhead camera was employed (held by a cohort from a staircase two floors above the testing area). The camera takes video of the test flight, where the aircraft is commanded to hover in a known location. Subsequently, the video is stabilized and then motion tracking is used to output the position of the aircraft in pixel coordinates for each frame. Finally, this is converted to physical units by using measured reference distances to convert pixels to feet.

Although this technique does not correct for probable error sources such as projection error, imperfections due to these sources are believed to be of small magnitude when compared to the signals being examined.

Fig. 4.27 and 4.28 show the ground-truth position of the aircraft during the test flight. They show a mean position error of 0.2 ft, or 2.4 inches, and a maximum error of 0.55 ft, or 6.6 inches.

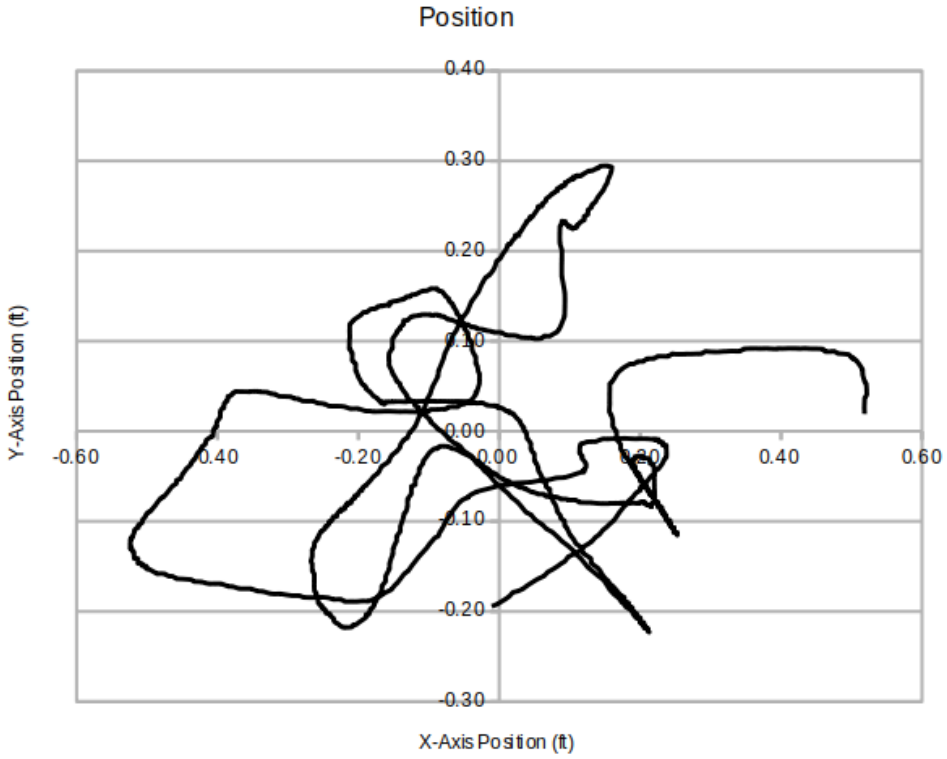

FIGURE 4.27: Position tracked during the test flight

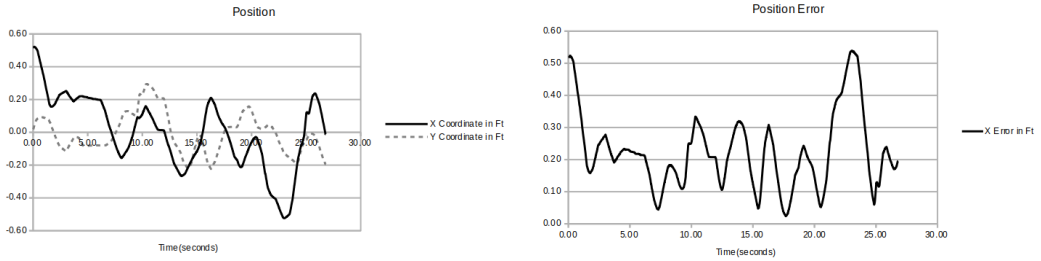

FIGURE 4.28: X- and Y-Position traces, and the net error distance from the (0, 0) setpoint

Fig. 4.29 and 4.30 show similar visualizations of the velocity (defined as the derivative of position over time). Measurements show significant noise due to the limitations of this video-based data collection approach. The average velocity error is 0.25 ft/sec, and the maximum recorded velocity error is 0.93 ft/sec.

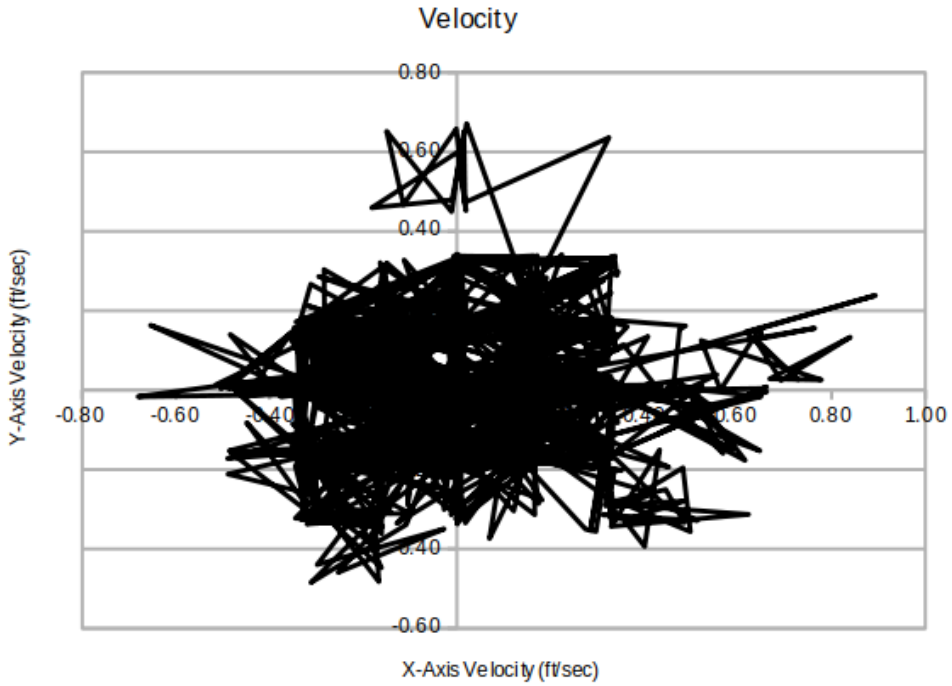

FIGURE 4.29: Velocity tracked during the test flight

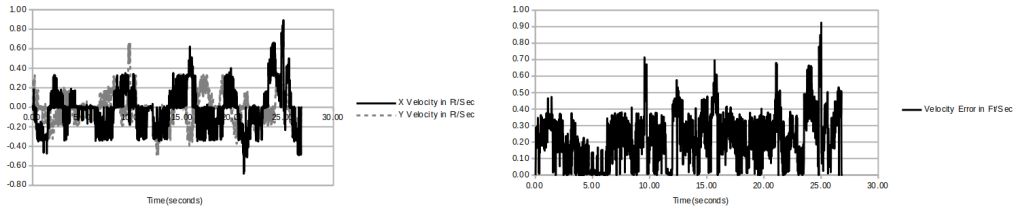

FIGURE 4.30: X- and Y-Velocity traces, and the net error distance from the (0, 0) setpoint

These measurements indicate that position holding accuracy is lagging behind position measurement accuracy, suggesting that more performance is available through improvements to the drone's position-holding ability. A nearby observer can clearly see the drone constantly hunting for position, though a more distant observer probably perceives the drone as still. The absolute error of typically 2.4 inches is encouraging, when viewed with respect to requirements for typical indoor applications.

For reference, Fig. 4.31 shows a sample frame of the video captured and analyzed for this section. The red square indicates the point being motion-tracked.

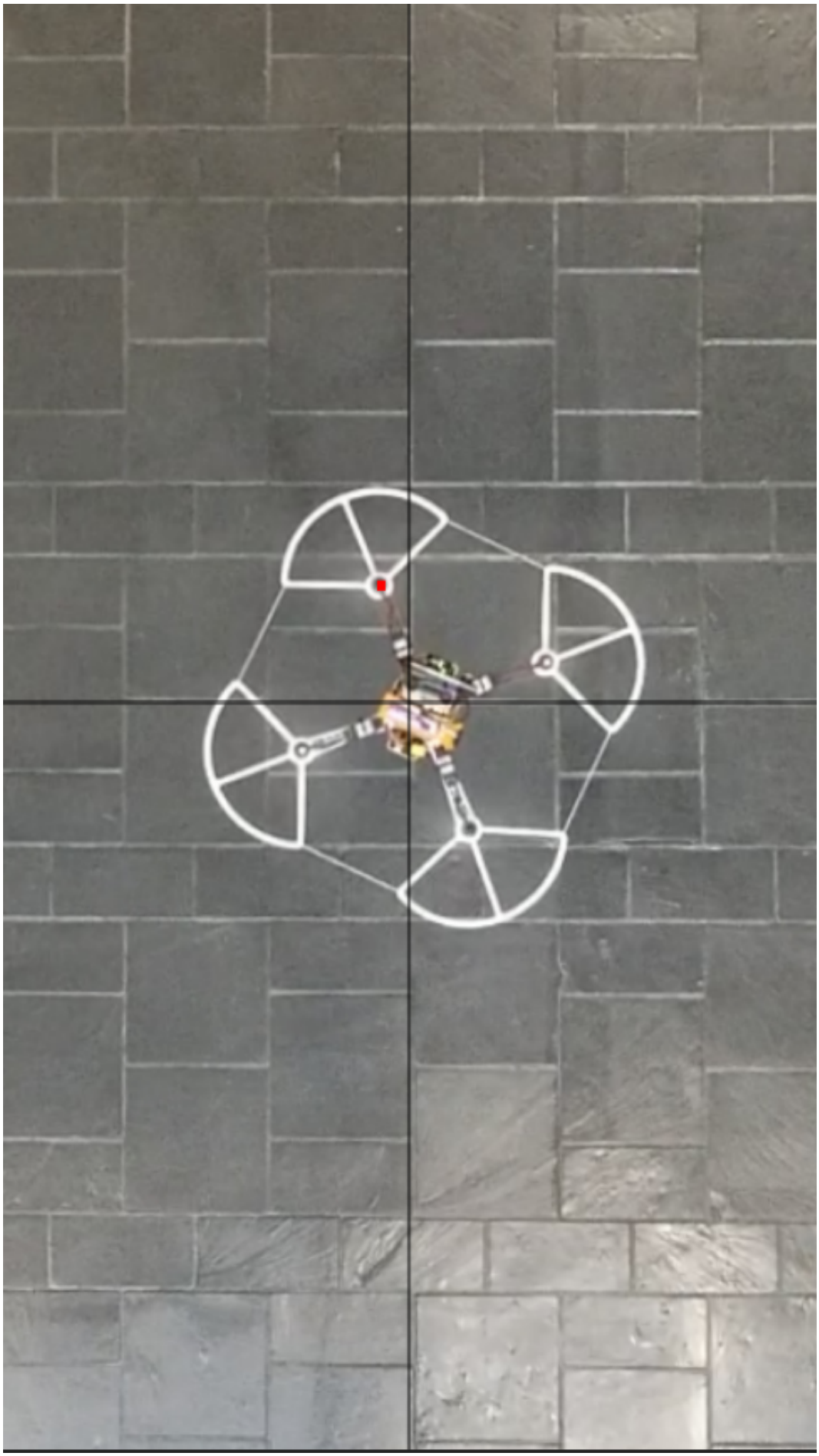

FIGURE 4.31: Image from video captured for position hold analysis.

Appendix C shows comparable data from a UBlox M8N GPS receiver operating outdoors.

4.9.2 Behavioral Testing

This section consists of "pilot reports" describing the performance of the aircraft and navigation system while operating in various modes and attempting various tasks.

Altitude Hold

Altitude Hold works well. Despite its name, movement is allowed in all dimensions in this mode, with some computer assistance. On the horizontal plane, the pilot's control stick translates directly into body pitch and roll, which in turn creates horizontal thrust and moves the aircraft. There is no tendency to stop or hold position. Vertically, the autopilot system controls the aircraft throttle with a feedback loop, and maintains a vertical velocity specified by the pilot, which can be either zero, positive or negative. This mode tends to feel like "drifting on a sheet of ice" due to its tightly-controlled vertical motion and momentum-conserving horizontal motion.

Unlike outdoor flight, there is zero drift as long as navigation markers are in view. During a sustained loss of navigation markers, altitude can drift slowly, and may "snap" back to position when markers are recovered, possibly surprising the pilot. In general the aircraft is both highly agile and highly controllable in this mode.

Position Hold

Position Hold mode takes the vertical control from Altitude Mode, and adds in horizontal control. In this mode, the pilot's joystick controls the velocity setpoint for horizontal motion (in the aircraft's body frame; axis are Front and Right). This mode does actively hold position when set to a velocity of zero. This mode engages additional control loops, and so relies on good horizontal velocity and position data for smooth operation. These control loops manipulate the aircraft's pitch and roll targets in order to create horizontal thrust when needed.

Flight in this mode is reliable and relatively smooth using the current settings. It is also very slow, and rocks back and forth slightly. The upper end of the aircraft's

tuning for responsiveness seems to be pretty low at the moment, and so low velocity and acceleration settings were chosen to prevent operation of the aircraft in performance regions that may lead to instability.

Waypoint Modes: Direct

This mode moves in a direct line to a waypoint, ignoring both pre-defined and discovered obstacles along the way. Since emergency collision avoidance is active, the drone still avoids flying into objects in directions where it has depth sensing, however this is currently only in the forward direction, meaning that the aircraft is at risk of side, rear, top, and bottom collisions. This mode is used to test the automatic position-seeking systems without inteference from obstacle avoidance. It could also be used when flying in close proximity to obstacles, such that the automatically obstacle discovery system would declare the current flight path "closed" and find an alternate route. This mode works as expected.

Waypoint Modes: Architecture

Architecture mode automatically generates multi-segment paths to avoid predefined obstacles, such as walls, doors, and tables. It has a predefined clearance distance of 2 ft, so it flies safely around obstacles in all directions and headings as long as it has visibility of an AprilTag to enable good position tracking. This clearance distance does limit maneuverability in tight spaces, where a human pilot could fit the aircraft through tight gaps.

Waypoint Modes: Obstacle Avoidance

Obstacle Avoidance mode is the most sophisticated navigation mode. In addition to planning around known obstacles, it senses new obstacles using its depth sensor, and then routes around them. The route can, and often does, change many times mid-flight, so as a pilot one can spend a lot of flight time wondering "what is it doing right now". However a glance at the GUI generally shows that the aircraft is correctly discovering obstacles and planning new routes, and once it's environment map "settles" it will have correctly determined a safe route to its destination.

A downside of this mode is that environment detection is somewhat slow, leading to the aircraft continuing forward towards a blockage for longer that should be necessary, before "seeing" it and changing direction.

The paths generated by this mode tend to be near, but not on, obstacles. This is a consequence of the path-generation algorithm, which tends to go right around the edge of obstacles when finding the shortest route to a destination. This presents a risky flight, and may be an avenue for future research.

Heading Mode: Direct

Heading Modes are a complimentary feature to Waypoint Modes: Waypoint Modes determine the path taken to reach a destination, while Heading Modes determine the vehicles heading while traversing that path.

Direct Heading Mode is the simplest: the user sets the aircraft heading directly and in real-time. Since there's no inherent relationship between heading and direction of travel, this can be used for purposes unrelated to navigation. For example, the aircraft could be rotated to aim a sensor at an area of interest, or it could be rotated into the orientation that's most natural for the pilot. There are also options that could be useful for navigation, such as orienting the aircraft such that it has maximum visibility of upcoming obstacles or of AprilTag fiducial markers.

In testing, this mode works well, though the pilot does take on the responsibility of making sure that the aircraft doesn't lose track of it's position. The aircraft rotates smoothly and accurately when commanded to.

Heading Mode: Waypoint

Waypoint Heading Mode keeps the aircraft pointed at it's next immediate waypoint. Immediate waypoints are the points generated by the path-generation system, rather than the user-specified ultimate waypoint. This mode is familiar to outdoor drone pilots, where drones are typically programmed to move in this manner.

This mode can result in frequent heading changes, especially when the aircraft replans its route in mid-flight. For example, if the aircraft discovers an obstacle and has to "back up" by a foot or two before continuing around the obstacle, this heading mode will command the aircraft to rotate 180°, move the one-to-two feet, and then rotate another 180° to move forward again.

Once this mode arrives at the final waypoint, it automatically reverts to Direct Mode, allowing the user to rotate the aircraft.

Heading Mode: Visual Markers

This mode is perhaps the safest mode for automatic navigation. In Marker Heading Mode, the aircraft continually identifies the nearest visual marker than would be expected to be visible to the aircraft, and turns the aircraft to face that marker. As the aircraft progresses along its route, it will turn to face different markers.

In practice this mode works well, and produces the best position accuracy during the length of the aircraft's route, as navigation "blind spots" are minimized.

4.10 Degraded-Mode Testing

This section describes testing done to quantify the effects of certain common adverse performance conditions. These can be distinguished from "failure" modes through the observation that the vehicle can generally continue operating through and beyond these situations, though possibly with reduced performance or operator interaction required. Each adverse performance mode is described, and then tests are executed to demonstrate the mode and quantify its effects on the aircraft, and finally recommendations are provided for operator actions to mitigate the effects of those conditions, and/or future research to eliminate the causal condition.

4.10.1 No AprilTags Visible

Perhaps the most common mode of degraded operation is the situation where no AprilTags are visible or detected. This is a common condition, since it can be caused purely due to the drone's position and attitude in space, and also by blockages or shadows over AprilTags. Note that AprilTags are particularly sensitive to the visibility of their corners and outer edges. Furthermore, a no-AprilTag-data condition can be caused by failures of the AprilTag detection algorithm. In extreme cases AprilTags can even be detected but discarded if their resulting attitude information disagrees with the current aircraft state.

Under this condition, the system still has visual velocity and rotation measurement. It should be able to maintain velocity and rotation stability. Position, altitude, and heading will drift slowly from their setpoints, but the magnitude of this drift is unbounded and will increase with time.

During the time where the drone is operating in this mode, it is highly susceptible to serious stability problems if a ZED Camera reset occurs. The "ZED Camera Reset" is a phenomenon observed where the ZED Camera's Point and Pose data streams freeze for 1-2 seconds, even though images are still being captured. It is assumed that a software reset of some kind is happening during this time period. ZED Camera Resets tend to be associated with sudden movements or whole-image changes.

Tests for this degradation condition focused on determining the flight-worthiness

of an aircraft while in this mode - both under fully autonomous navigation, and under operator control. Software was created for the aircraft that disables the AprilTag detector for periods of time, with the period ranging from 0.25 seconds to 32 seconds. At the end of the data-loss period, the position delta between the first new position measurement, and the internal position state estimate is calculated. This number reveals how far the position estimate has drifted from the real position. Additionally, pilot notes are kept for any unusual aircraft behaviors. Four iterations of this test are performed, each with differing flight patterns: Still, slow right-left translation, fast multi-directional translation, and slow clockwise-counterclockwise rotation.

Results from this test, which are displayed in Figures 4.32, 4.33, 4.34, and 4.35, differed from expectations in that there was only a vague relationship between loss-of-data time and error magnitude. One one hand this is encouraging, in that the does demonstrates that a vehicle can run for 30+ seconds with no position data, and still maintain an accurate position estimate. It is hypothesized that the ZED Camera's velocity measurements are based on an internal SLAM (Simultaneous Localization and Mapping) system, which is immune from drift as long as reference points are kept in view. Therefore it may be the case that the signal in this experiment is below the noise level. Since this experiment compares AprilTag measurements against the state estimate, the noise levels of both of those values are relevant. AprilTag measurements in particular have noise proportional to the squared of the distance from the AprilTag, which may explain why the values seen for very small loss durations (0.25 seconds, 0.5 seconds) is still significant.

Delay	Trial	Position Delta
0.25	0	0
0.5	0	162.79
1	0	127.53
2	0	26.589
4	0	149.5
8	0	107.492
16	0	265.95
32	0	160.253
0.25	1	85.7316
0.5	1	204.41
1	1	251.71
2	1	173.18
4	1	34.1535
8	1	35.5468
16	1	10.4475
32	1	109.804
0.25	2	14.3664
0.5	2	10.8368
1	2	55.883
2	2	26.025

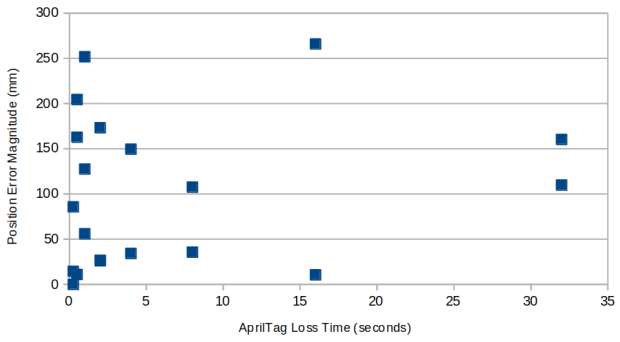

FIGURE 4.32: Position error magnitude after temporary AprilTag removal. Drone in a steady hover. Under these testing conditions, there seems to be no correlation between position error and data dropout duration.

Delay	Trial	Position Delta
0.25	0	0
0.5	0	6.03096
1	0	31.1691
2	0	138.101
4	0	177.041
8	0	355.789
16	0	68.8141
32	0	259.444
0.25	1	303.978
0.5	1	141.971
1	1	153.559
2	1	35.1424
4	1	301.265
8	1	87.5187
16	1	288.776
32	1	354.814
0.25	2	43.2141
0.5	2	79.8654
1	2	108.596
2	2	57.4572
4	2	89.4632
8	2	262.356
16	2	244.905
32	2	278.943
0.25	2	35.5871

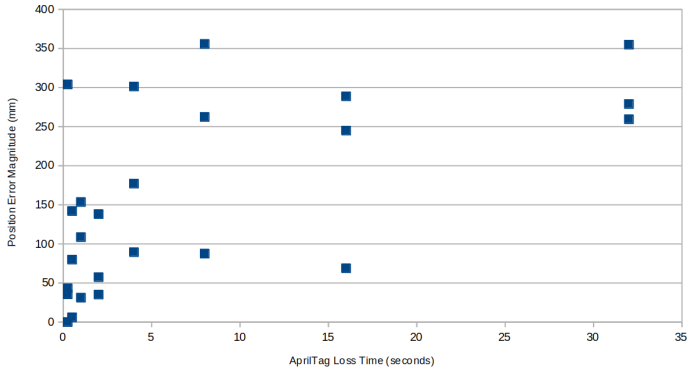

FIGURE 4.33: Position error magnitude after temporary AprilTag removal. Drone moving slowly left-right. When moving, there seems to be a weak correlation between data dropout duration and position error.

Delay	Trial	Position Delta
0.25	0	0
0.5	0	34.0961
1	0	239.63
2	0	639.271
4	0	422.208
8	0	571.686
16	0	756.066
32	0	1455.7
0.25	1	181.97
0.5	1	164.209
1	1	308.893
2	1	407.942
4	1	666.962
8	1	770.651
16	1	991.771
32	1	3033.2
0.25	2	107.267
0.5	2	505.099
1	2	868.423
2	2	704.517
4	2	280.003
8	2	1293.62

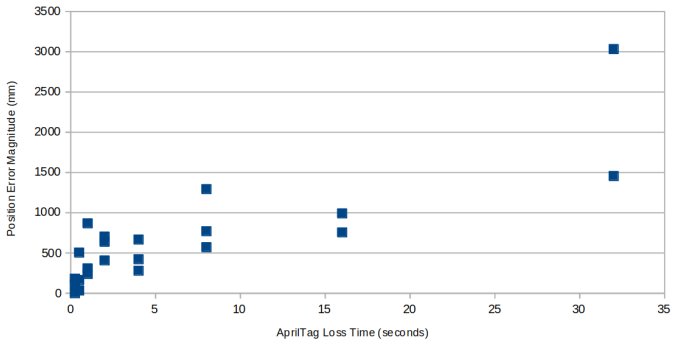

FIGURE 4.34: Position error magnitude after temporary AprilTag removal. Drone moving quickly and erratically on the translation front and right axis. Under these flight conditions position error generally grows the longer that data is lost.

Delay	Trial	Position Delta
0.25	1	0
0.5	1	74.1558
1	1	42.5219
2	1	74.1854
4	1	49.0572
8	1	119.399
16	1	243.818
32	1	177.432
0.25	2	48.8935
0.5	2	76.0855
1	2	81.6433
2	2	127.09
4	2	102.26
8	2	52.98
16	2	164.635
32	2	113.254
0.25	3	33.069
0.5	3	52.4425

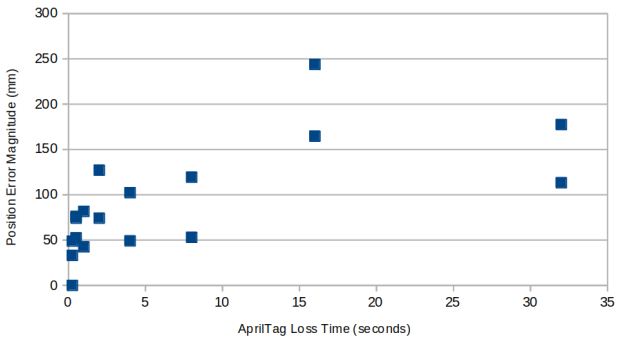

FIGURE 4.35: Position error magnitude after temporary AprilTag removal. Drone rotating on yaw axis clockwise and counter-clockwise. This test uses rotation motion rather than translational, but obtains a similar result with the position estimate tending to degrade in the presence of longer data dropouts.

4.10.2 Complete Video Loss

Taking the loss-of-data concept to the extreme, a complete loss of usable video is a possible event, even with a fully functional system. Such an event could be caused by extreme lighting changes, which cause the image to "white out" or "black out" until the exposure algorithm has had time to compensate. Such an event could also be caused by debris obscuring the camera's field-of-view, or dust or fog limiting visibility.

Without visual data, not only is the absolute position reference lost, as tested in Section 4.10.1, but horizontal and vertical velocity measurements are lost as well, forcing the flight controller to dead-recon from IMU measurements only. The Kalman filter and related state estimation code in the flight controller operate in this dead-reckoning mode for up to 10 seconds. After 10 seconds without position or velocity data, the state estimator will reverting to "constant position mode", which is incompatible with position- and/or altitude-controlled flight.

Due to the potential for loss of control during flight, this experiment is done in a lab setting, with the vehicle fixed to a motion-controlled sled. The software system aboard the vehicle is modified to periodically block all inputs from the state estimator for variable periods of time (ranging from 0.25 seconds to 32 seconds), and then compute the velocity error after data is resumed. This is done both with the aircraft station, and with the aircraft moving horizontally back-and-forth during the data blackout.

During video dropouts the state estimators variance estimate for position and velocity signals can be seen to increase exponentially, as one would expect during a sustained loss of information. Fig. 4.36 shows this effect, along with the mean values for a position and velocity axis. Although the direction (positive or negative) of the drift should be random, one would expect the mean values to drift at an increasing rate over time. Under this degraded operating condition, velocity is only being updated with the integral of the accelerometer's measured acceleration, and position is only being updated with the second integral of the same.

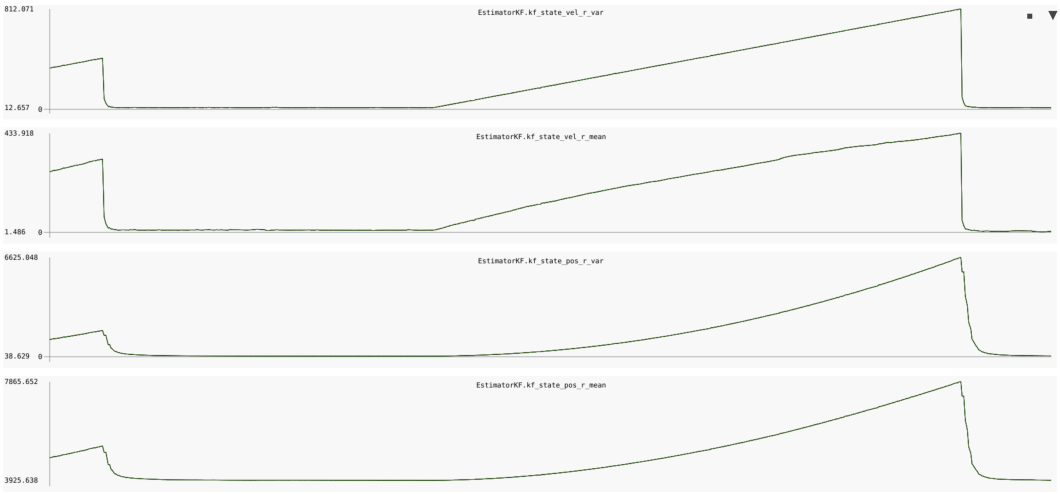

FIGURE 4.36: Velocity Mean and Velocity Variance traces. The sawtooth-shaped segment is the data dropout period. These traces show that without any data sources beyond accelerometers, the mean accumulates error quickly and without bound, though the variance does accurately reflect this loss of precision.

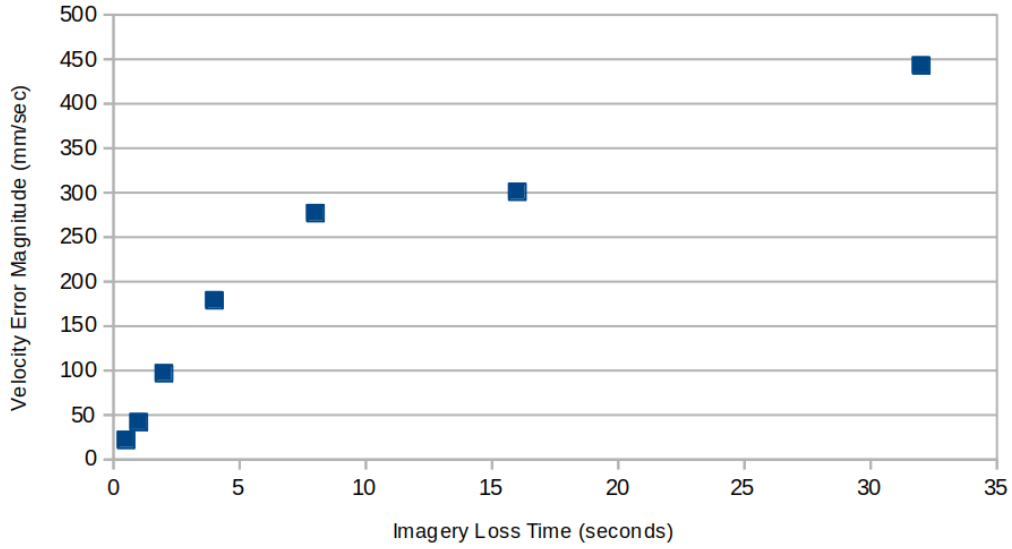

FIGURE 4.37: Velocity estimate error, which affects the aircraft's ability to hold position and altitude, increases with the duration of a video loss.

By varying the data loss periods from 0.5 seconds to 32 seconds, the following graph (Fig. 4.37) was obtained, showing that velocity error grows in relation to the error duration.

The aircraft's altitude control is primarily a feedback loop based on the aircraft's estimated vertical velocity, and the aircraft's position control is a pair of feedback loops based on the aircraft's estimated horizontal velocities. Thus, any systematic

error in velocity estimates is turned into an equivalent amount of motion in the aircraft. Other side-effects, such as badly biasing the accelerometers and gyroscopes are possible as well, leading to a loss of attitude control. In short, if this condition persists for more than a few seconds, position and altitude control should be disabled, and the aircraft should be landed through manual piloting as quickly as possible.

Based on these results, the final moving test was skipped, as it is already safe to conclude from this best-case failure testing that this type of error quickly leads to a loss of position and altitude control.

4.10.3 CPU Overload

The system described in this paper has large computation demands. Between the ZED Camera's SLAM system, the full-frame AprilTag detector (possibly running on multiple cameras), and the other algorithms and communication channels, it cannot be taken as a given that a low-power CPU can keep up with the real-time demands of the system. There is some flexibility within the system to control its CPU load, as a trade-off against performance. For example, lowering the resolution of the cameras significantly decreases the demand for computational resources, at the cost of lowering the maximum distance at which AprilTags can be resolved. Similarly, the frame rate of the cameras can be lowered, at the cost of increasing the system latency, which may require softening of the subsequent control loops.

Since the visual navigation system's computing platform is a Linux computer, the system must be capable of operating in an environment where it doesn't have exclusive use of the CPU, GPU, RAM, and I/O resources. Even a minimalist typical Linux system still has 100+ processes running, which cause unpredictable load patterns.

This discussion should be taken to suggest a need to understand the CPU usage and CPU performance of the current system. Furthermore, experiments should be performed to understand the extent to which CPU performance degrades when placed under additional load.

Several key factors were identified as representative of resource utilization on the processing system:

- CPU Usage - As reported by the "top" command, this metric shows the amount of compute time that is used, compared to the amount of compute time available. This is normalized to the capacity of one CPU core, so one fully-loaded core reads as "100%", while two fully-loaded cores reads as "200%". The Jetson TX2 has six cores, so it has a theoretical maximum of 600% usage.

- Load Average - This metric shows the average length of operating system task scheduler's run queue. A reading of "1.0" means that there is on average 1 task requesting CPU time at any given moment, which is roughly similar to 100% CPU Usage. Since tasks may have time-sensitive demands for compute resources, there can be moments in time where several tasks are competing for CPU time, even if the overall CPU usage is less than 100%. This metric helps identify the amount of competition for CPU time.

- ZED Camera Frame Rate - The frame rate at which the ZED Camera is able to operate, including the signficant CPU and GPU loads involved in post-processing each frame to track motion and compute depth. This should match the configured frame rate with no jitter; any lower number indicates that the ZED driver is starved for resources.

- AprilTag Detector Frame Rate - The rate at which the AprilTag detector is able to fully process images and emit AprilTag detections. This should normally be equal to the ZED Camera rate, as the ZED Camera is the source of imagery for the AprilTag detector. Any lower number indicates that the AprilTag detector is starved for CPU time.

- Aries Output Frame Rate - The rate at which position and velocity messages are emitted from this system towards the Aries flight controller. This is specified to be 30 FPS, and requires few resources to do. It is included as a "catastrophic failure" indicator, as any deviation from the target message rate would indicate major resource starvation, essentially putting the visual navigation system in an unusable state.

- FixedMapNavPoints Time - This is a fixed-size computation task that is repeated periodically, using whatever spare CPU time is available. Longer times indicate that few spare CPU cycles are available.

Measurements of these factors are taken both with the system running normally, and with an additional synthetic load added. The synthetic load is three processes running "cat /dev/urandom > /dev/null". Each process is single-threaded and CPU-bound, and so should attempt to consume 1 CPU core. The results are such measurements are displayed in Table 4.8.

Metric	**Value - Normal**	**Value - Synthetic Load**
CPU Usage (%)	395	298
Load Average	5.01	7.64
Memory Usage (MB)	487	487
ZED Camera Frame Rate (FPS)	30	19
AprilTag Detector Frame Rate (FPS)	30	19
Aries Output Frame Rate (FPS)	30	30
FixedMapNavPoints Time (s)	3.9	8.5

TABLE 4.8: Measured deviations from angle setpoint, as calculated by the aircraft state estimator. Aircraft hovering in ground effect at 6" above floor

These results show both evidence of good health under normal load, and poor health under additional synthetic load. Under a normal load, the ZED Camera and AprilTag frame rates are stable at their target rate of 30 FPS, indicating that the system is able to keep up with imagery processing at the rate that images are generated. Not shown, but also available is information about the rates of many other internal processes, which are performing at their expected rates. Under the synthetic load, the Load Average increases by approximately the number of extra processes added to compete for CPU time. The CPU Usage actually decreases, but this is misleading: the system is now dropping frames, so there is less total data to be processed.[1] The key factors for identifying system load problems are the frame rates: neither

[1]In the experiment that was the subject of this discussion, a two-core synthetic load was created. This resulted in the system dropping every other frame, essentially halving the overall computational requirements. One a 6-core Jetson TX2, the net effect was a drop in load average.

the ZED Camera nor the AprilTag detector are able to run at their specified frame rate. Finally, the FixedMapNavPoints algorithm takes much longer to run, since it's in stiffer competition for CPU time.

Chapter 5

Testing and Operation Notes

This chapter is meant to be a set of best practices and lessons learned with respect to setting up and operating an aircraft with the Visual Navigation System. As a research project, the system is far from turnkey, but works well when properly configured and used within its limits. The initial set-up and maiden flights of a new aircraft are high-risk operations, and should be undertaken slowly and deliberately, with an understanding of the best order to activate, tune, and test subsystems in.

Since the demonstration system uses an Aries flight controller, this section also exists as an opportunity for the author to document procedures and settings for the configuration of an Aries-based multirotor.

5.1 New Aries Multirotor Setup

Note that the following discussion assumes standardized sensors, therefore it is most applicable to outdoor drones utilizing GPS receivers and magnetometers. It assumes that all electronics are wired correctly and working as intended. Steps are similar for indoor drones, however the tuning of Altitude and Position controls must be interleaved with testing of the indoor positioning system, to ensure that data published to the flight controller is correct.

Assuming that the aircraft is not identical to some existing aircraft for which Aries settings already exist, the user must turn the PID controllers and Motion Control parameters to achieve stable flight and reasonable movement. The flight control is designed with hierarchical control loops, such that high-level loops depend on low-level loops. Due to this design, low-level loops should be turned first, and testing should be done in such a way that only the necessary controllers are running. Fig. 5.1 shows the approximate layout of the PID controllers and Motion controllers.

Setting up a new multirotor starts with choosing some initial guesses at parameters. The built-in ones are not usable. It's best to choose parameters from a similar aircraft, and then de-tune (lower) them. Starting completely from scratch is a difficult option, since even figuring out of the correct order of magnitude for the parameters can be difficult, and the aircraft may be unflyable as this is happening. The normal method of PID tuning used at the VCU UAV lab is to increasing each parameter until it induces oscillation, and then reduce the parameter until oscillation

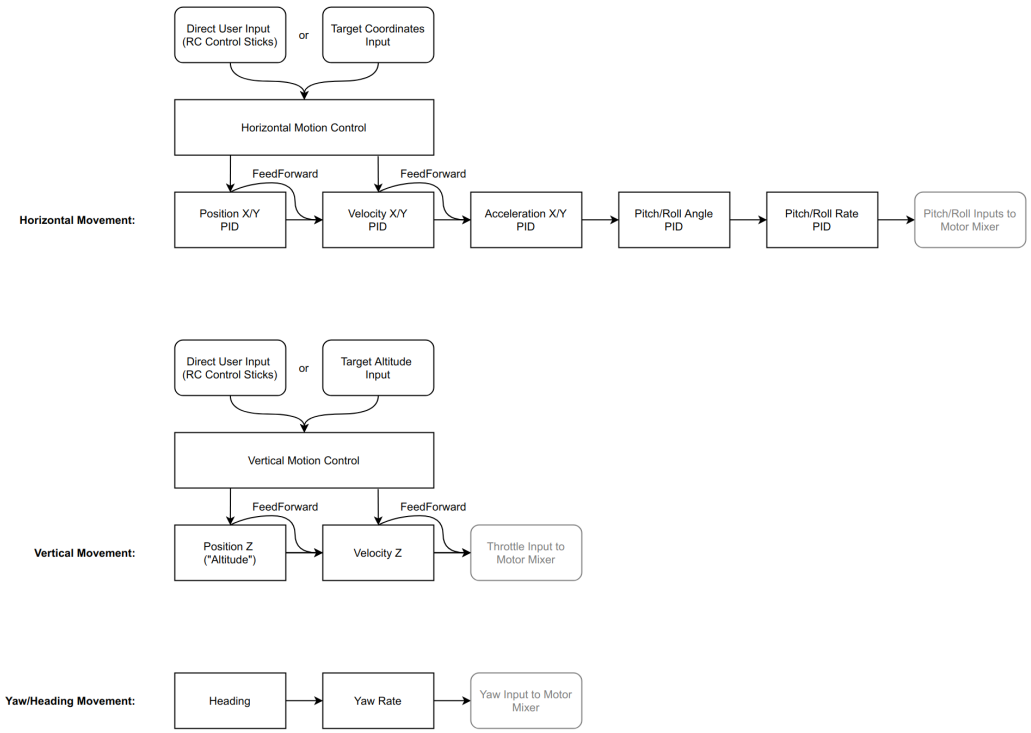

FIGURE 5.1: Controller hierarchy in the Aries Flight Control System

stops. Parameters are normally done in this order: P, D, I. This isn't an exact science though, as filters must be adjusted at the same time to limit the amount of noise passed into the controllers (at the expense of increased latency), and compromises must sometimes be made to this tuning sequence just to get the aircraft to be capable of flight without crashing immediately.

Before each controller is tuned, one should identify the "limit" parameters related to that controller: parameters that limit the maximum values or rates that the control can command. For example, the Angle mode controller has settings for maximum pitch/roll angle, maximum rotation rate and rotational acceleration, and maximum yaw rate and acceleration. Particularly for the higher levels of automation, such as altitude and position control, these limits should be set to unusually low values initially, and then widened to normal limits as confidence in the system increases.

Caveats out of the way, the first pass at the aircraft should be focused on the rate-mode pitch, roll, and yaw controllers. Fly in Angle mode, with a low P gain (1 or 2), and zero I and D gains on the Angle controllers. The tune the Rate-mode P, I, and D controllers by increasing each one until sinusoidal oscillation occurs (not random motion, but definite sine waves), and then back them off until the oscillation stops

and the motors sound smooth.

Rate-mode tuning is the hardest part of the process, requiring the most iterations, and the most careful test piloting. As a test pilot, be prepared for an unstable aircraft. First test that disarming and manual safeties work. Then see what testing can be done with the aircraft running, but with the throttle low enough that it can't take off. This may be enough to determine that the aircraft learns in the correct directions in response to the pitch/roll stick, and may be enough to detect some oscillations before getting into free, but very unstable, flight.

The Angle mode configuration always results in a full set of parameters for the Rate controllers (P, I, D, I-Limit, and Filters), and only a P Gain and P Filter for the Angle controller, with all other parameters being zero. This is a pattern amongst cascaded PID controllers, where it is often the case that the lowest-level controller is fully-tuned, while more senior controllers are a proportional gain only. There is also a predictable decrease in speed or responsiveness that comes with each additional level in the hierarchy: low-level controllers are the fastest, quickest-reacting controllers, while higher-level controllers become successively softer and more latent.

After tuning Rate mode (and getting Angle mode for free with it), tune the Altitude mode. Again, start by copying and then detuning existing PIDs. Again, there should be a P gain only in the upper ("position") loop, and a full set of parameters for the middle ("velocity") loop. The Altitude controller introduces a third ("acceleration") controller, but this is normally not used and all parameters should be set to zero. Under the Altitude Settings section, set appropriate values for Hover Throttle and for the acceleration and velocity limits, and make sure both feedforwards are set to 1.0. Altitude mode can be tuned with the upper controller P-gain set to zero while the middle controller is tuned. After the middle controller is tuned for maximum vertical stability, and the upper-level P-gain can be experimented with to give altitude holding based on the measured altitude value, rather than the vertical velocity values.

Finally, Position PIDs are similar and are tuned similarly to the Altitude PIDs.

5.2 Indoor Navigation Module Setup

The Indoor Navigation Module is a Jetson TX2 computer, ZED Mini Stereo Camera, and a UART connection to the Aries flight controller. After assembling and wiring the system, the base Jetson image, which contains the Linux operating system and nVidia CUDA drivers, should be installed according to nVidia's instructions. VCU users can also use the detailed instructions stored in the UAV Wiki server.

Next, install the ZED libraries from StereoLabs, using their instructions for installation on a Jetson.

Next, install the Visual Navigation system by cloning the repository from VCU's servers. Once cloned, there is a "build and run" script in the build directory. The remaining steps consist of installing dependencies until the build completes successfully. Details for this are beyond the scope of this document, but can be found on the VCU UAV wiki.

A final note is that a running Aries must be connected to the Visual Navigation system in order for it to fully start up. Since the system identifies its communications based on the aircraft's tail number, it waits (potentially forever) to receive information about the aircraft's tail number from the Aries FCS before finishing initialization.

Chapter 6

Conclusions and Future Work

6.1 Conclusion

Drones are increasingly making their way into everyday commercial and industrial use, and with this growth comes the challenge of operating in constrained environments. This body of research demonstrates that safe, precise operation is possible within complex indoor or warehouse environments, but that such operation is not a simple adaptation of outdoor technologies to indoor environments, but must be built to purpose. The need for precise localization, beyond the accuracy of standard GPS systems, is highly compatible with the array of sensors available for close-range operation, while the need for environmental mapping and sensing dovetails with emerging technologies in computer vision, machine learning, and high-performance specialized compute hardware. These trends will continue, as hardware designers focus on low-power, high-performance specialized devices rather than general-purpose compute. And this is great news for the drone designer, as it ultimately means more power in a smaller footprint - both physical and energetic.

The demonstration system developed during this research is a fully-functional proof-of-concept. It achieves all of the goals proposed for the research not only in theory, but in practical demonstration. In particular, this research demonstrated:

- Highly-precise indoor localization based on optimal markers
- Dynamic path generation and updating using both known and unknown environmental details
- Automatic collision avoidance

These successful demonstrations show the feasibility of drones for indoors, close-quarters operation. Such a system could be particularly valuable to industrial plants, where flying robots can help to solve the "dull, dirty, or dangerous" problems described in the introduction to this paper.

Practical demonstration is seen by the UAV lab at VCU as unusually important when compared to many other research organizations, but for good reason: the control theory, state estimation theory, motion planning theory, and electrical and computer programming knowledge that constitute the raw ingredients of an

autopilot system are well-known and well-documented. Yet, autopilot and navigation systems are not settled. This gap between the known and the experimental is explained, in the author's opinion, by the reality that system performance is often bounded not by theory but by practical limitations: sensors have noise and failure modes; environments change over time; computing power, energy usage, weight, and form factor are all at odds with each other. Practical demonstration of an indoor autopilot system which can discover and re-route around obstacles, deal with large-volume environments, and maintain position estimates with centimeter-level accuracy is significant as an indicator of the level to which theory can be transformed to real utility.

Society may be on the edge of a "Drone 2.0" era, where the issues of flight and navigation and safety leave the forefront as accepted solutions gain widespread implementation. In their place, an exciting new era of development may be beginning with a focus on applications and widespread integration. Drones with polished APIs that interconnect with commodity hardware and networking have the potential to redefine the UAV research in the same way that personal computers redefined computer science, and smartphones redefined portable computing. For the researcher working in this field, they should be encouraged that their work is timely and socially relevant. Yet they must also aware that this is a time of change for UAV engineering, and so their focus must not be solely on the technical details and theoretical mathematics of any given problem, but on the wider commercial and societal trends that the problem exists within. Researchers equipped with an informed world view and first-class technical training have the ability to influence the future, and should be nervously excited about that possibility.

6.2 Future Work

Looking forward, this project opens the door to numerous follow-on projects. Many individual components - the path generation system, the visual odometry system, data fusion and state estimation, and data management for the environmental map - are ripe for taking as an area of concentrated study. These components exists in

the demonstration system as functional but not cutting-edge components, so a deep dive into current research trends could be applied to any of the components.

The algorithm for understanding the state of the environment by accumulating depth data into voxels could be improved or replaced. The speed at which this algorithm is able to internalize changes in the environment, without the introduction of excessive noise, becomes a limiting factor for how quickly drones can move through the environment. Additionally, decisions can be made about weather this capability should remain available only in the direct the drone is currently facing, or if it should use some combination of sensors to increase the drone's effective field-of-view.

Imagery is similarly a large part of the system, and the system is only beginning to scratch the surface of what can be done with real-time imagery, especially when combined with computer vision research. Using self-driving cars as an example, there are clear benefits to being able to identify signs, people, and other objects. Signs, for example, could be use to restrict the drone from entering certain areas without requiring that information to be entered digitally. Thus maintenance crews (again as an example) could put up warnings and signage for drones in the same manner as they do for human.

More work could certainly be done to increase the performance and capacity of the navigation points system, and the background task which tries all possible paths to see which ones are currently blocked. This could take multiple directions: either a straightforward focus on performance optimization, or a focus on minimizing the number of paths and points in the relevant datasets, with minimal impact on the ultimate routes created.

One could also look towards insertion of visual-navigation enabled drones into real industrial and commercial applications. This could be treated as a study to identify problems, needs, and strengths of drones in industrial or commercial applications based on actual experience. The time for this study is now, since drones in both research and commercial production are just now gaining the capabilities necessary for these operations. Such a study should focus on fleet management as well, keeping in mind that the drone becomes just one of many tools and machines that a business will need to run with minimal oversight and maintenance.

In developing a software architecture for the management of a three-dimensional

environment, one should keep in mind the complexity of this topic, and that fields are familiar with this problem and have developed tool-sets for approaching it. The Geographic Information Systems (GIS) field is experienced in managing multiple sets of disparate mapping-based data, for example.

The test system uses hand-written functions for 3D geometrical calculations. These are messy, and quite possibly buggy, as they are a subject requiring substantial mathematical expertise, and the author finds his lacking. A future version of this system should benefit from the confidence of a well-tested 3D geometry library. Open3D, LibIGL, and CGAL (the Computation Geometry Algorithms Library) are all minimalist libraries suitable to this approach, while game engines such as UnrealEngine and Unity are more prescriptive solutions, but also include useful capabilities such as dynamically loading in world data as the user moves.

This "Get to the waypoint" commander only scratches the surface of the opportunities available to drones with this level of sensory information. More complex Commanders could run exploratory routes to discover and update their internal maps; run hybrid user+automation movement algorithms for manual control with automatic obstacle avoidance; do videography-focused tasks like orbiting a target. They could implement automatic return-to-home functionality, or a group of drones could cooperatively accomplish a series of tasks while their commanders are in communication with each other.

From a software perspective, the Commander could be the starting port for an "App" interface, where 3rd-party modules are developed to accomplish these tasks, and "plugged in" to the drone via a public API to the Commander.

Appendix A

Additional AprilTag Testing

This test and all of the tests in this section are precursors to the formulae developed in section 3.2.3.

vs. Distance

An analysis of AprilTag position and pose estimate variance was undertaken. As in all experiments in the section, variance data points were generated from 300-sample datasets using translation and rotation data captured from the AprilTag API, and using a ZED camera as the imagery source. All experiments incorporated small amounts of vibration in the system to avoid the "frozen image" problem, where the camera captures identical images and generates statistically flawed data. Fig. A.1 shows the results of this experiment, which indicate that that variance does not depend on distance. Angular variance follows the same pattern.

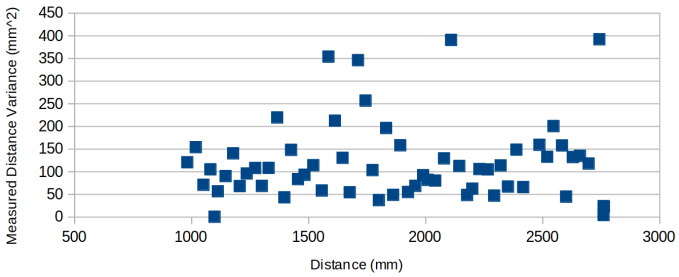

FIGURE A.1: AprilTag pose estimation distance variance vs. actual distance. This data suggests that there is no relationship between distance and measured variance.

vs. Resolution

Resolution is more straightforward - experiments show that increased resolution decreases variance, as shown in Fig. A.2. Again, angular variance follows the same pattern.

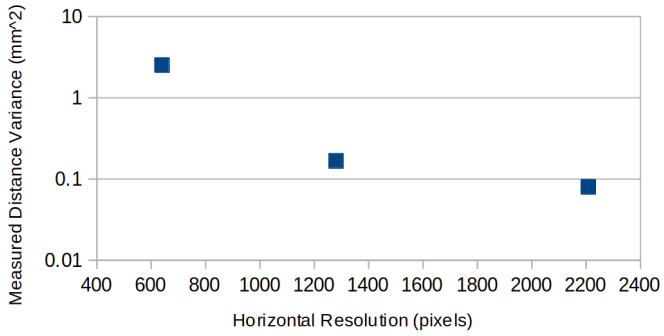

FIGURE A.2: AprilTag pose estimation distance variance vs. camera resolution. This data suggests that increasing camera resolution improves (decreases) variance.

vs. Angle

This experiment measures the effect of the targets rotation on the yaw axis relative to the camera. Note that there is a theoretical maximum of 90 degrees rotation before the target will become obscured from view of the camera. Angle variance displayed an unusual and unexpected result: variance increases strongly when viewing the target straight-on. Precision can be an order of magnitude better where viewing the target at an angle of 30 degrees. Fig. A.3 shows data collected to draw this conclusion. It is hypothesized based on observations that the two-solutions problem inherent to four-point, 2D to 3D correlation problems [79] manifest themselves more frequently is these orientations, resulting in solutions that randomly toggle between two values.

A.0.1 Variance Estimation

While these data aren't sufficient to fully describe the point-and-pose variance of an AprilTag system, they can be used to create guidelines. Since they show that variance is unaffected by distance, correlated with resolution and size, and uniquely related to viewing angle, an estimated variance can be computed if given a reference

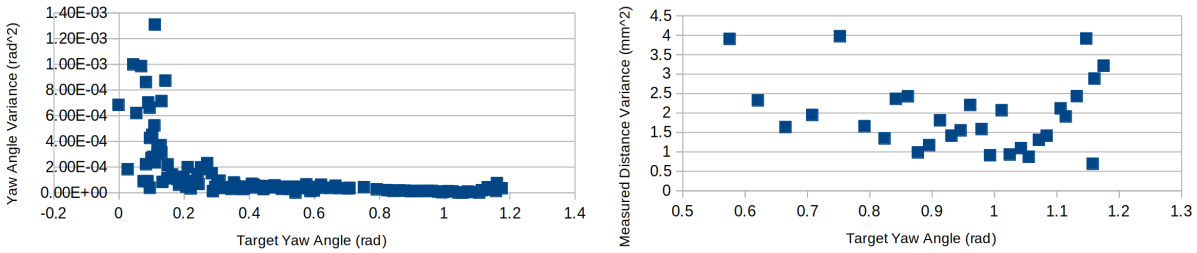

FIGURE A.3: AprilTag pose estimation distance and angle variance vs. target angle, where an angle of zero indicates that the target is parallel to the camera's image sensor. This shows an unexpected effect, where targets angles close to zero (i.e. fully visible; parallel to the camera's image sensor) have unusually large measured-angle errors.

variance measured under known conditions. With respected to measured variances $\sigma^2_{d,m}$ for distance and $\sigma^2_{a,m}$ for angle, at distance d_m, resolution r_m, size s_m and an angle of 45 degrees, the following formula gives the approximate expected variance of data taken from the same camera under different conditions:

$$\sigma_d^2 \approx \sigma^2_{d,m} \cdot \frac{r_m}{r} \cdot \frac{s_m}{s} \cdot 1.4 \cdot 10^{-5} \cdot (a - 0.785)^{-1.3} \tag{A.1}$$

$$\sigma_a^2 \approx \sigma^2_{a,m} \cdot \frac{r_m}{r} \cdot \frac{s_m}{s} \tag{A.2}$$

Appendix B

Test Vehicle Angular Stability Testing

A multicopter is never absolutely still in-flight. It is in an unending cycle of toppling and catching itself. The magnitude of unplanned pitch, roll, and yaw deviations can be a useful indicator of the degree to which the drone can regulate its own attitude. This becomes relevant to the Visual Navigation system when that system is attached to the drone, such that all drone rotations become camera rotations as well. For although the visual navigation system does determine and use its actual pitch, roll, and yaw for all attitude, velocity, and position calculations, these estimates are imperfect, and an easy way to improve them is to eliminate unwanted movements from the aircraft.

This experiment is similar to the Vibration test, in that the aircraft is hovered in and out of ground effect, while data is collected. In this experiment "Altitude Hold" mode is used, rather than "Position Hold" mode, so that the pitch, roll, and yaw rate targets can be easily fixed at zero. The data collected for this experiment is the flight controller's pitch, roll, and yaw estimates, as well as pitch, roll, and yaw rates. RMS error calculations can then be performed. Data is collected at 50 Hz.

The following tables B.1 and B.2 present the RMS mean angular errors and angular rate errors measured during this experiment:

Axis	**RMS Angular Error Magnitude**
Roll Rate	5.2 deg/sec
Pitch Rate	7.5 deg/sec
Yaw Rate	2.4 deg/sec
Roll	0.7 deg
Pitch	0.6 deg
Yaw	n/a

TABLE B.1: Measured deviations from angle setpoint, as calculated by the aircraft state estimator. Aircraft hovering in ground effect at 6" above floor

Axis	**RMS Angular Error Magnitude**
Roll Rate	4.5 deg/sec
Pitch Rate	7.2 deg/sec
Yaw Rate	2.3 deg/sec
Roll	0.5 deg
Pitch	0.4 deg
Yaw	n/a

TABLE B.2: Measured deviations from angle setpoint, as calculated by the aircraft state estimator. Aircraft hovering in clean air at 6' above floor

Appendix C

GPS Position Data Measurement

In order to contrast the precision of the Visual Navigation System against traditional UAV hardware, data was collected from a UBlox M8N GPS receiver. By allowing the receiver to sit in a stationary position under ideal conditions, the measured positions reported by the receiver could be plotted against time, establishing the receiver's precision.

Data was acquired from the GPS receiver using NMEA sentences, and recorded as 32-bit floating-point latitude and longitudes. The limited resolution of this numeric format when applied to a worldwide coordinate system explains the banding visible in the data: the least-significant bit of data represents a movement of approximately 2 feet. These data are plotted in C.1 and C.2.

These data can be compared to similar VNS experiments described in 4.9 and illustrated in 4.27 and 4.28. One can conclude that the outdoor UBlox GPS system has an average position error of approximately 4 feet. The theoretical precision of a single (non-differential / non-realtime-kinematic) GPS receiver is predicted in the yearly report, "An Analysis of Global Positioning System (GPS) Standard Positioning Service (SPS) Performance" by the University of Texas at Austin. Their worldwide average Position Dilution of Precision (PDOP) value, which represents the typical error to be expected in position solutions, is 1.62m or 5.31ft [92]. The VNS, by contrast, demonstrates in-flight position holding accuracy (a harder problem) of 2.4 inches.

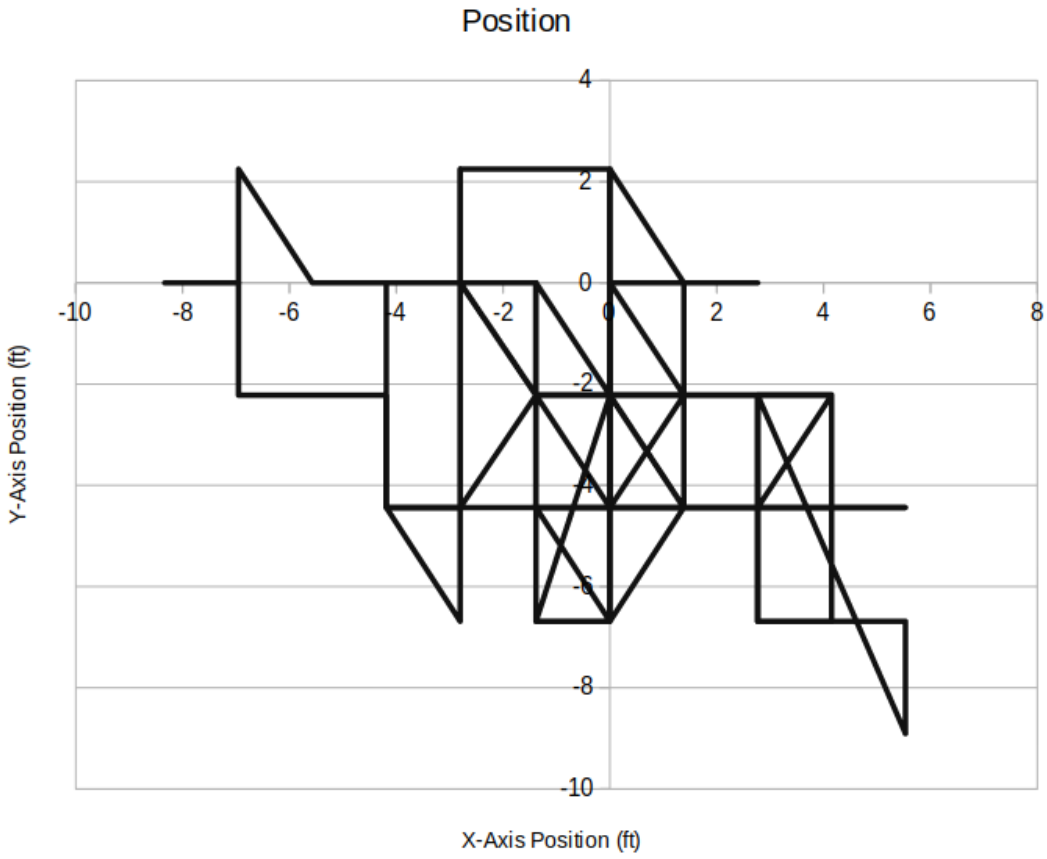

FIGURE C.1: GPS reported position while sitting motionless on the ground under favorable conditions.

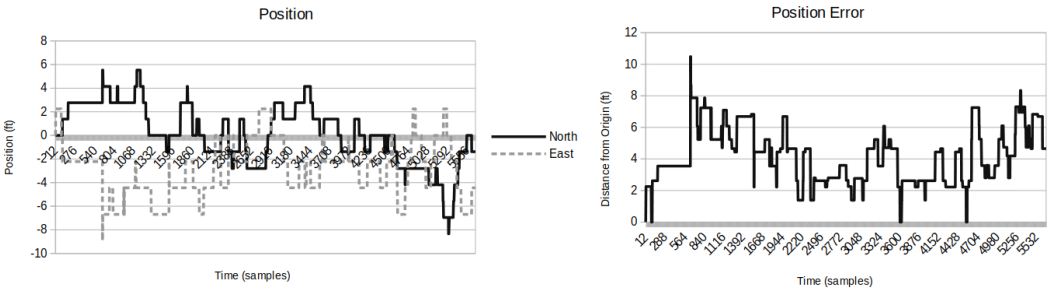

FIGURE C.2: GPS X- and Y-Position traces, and the net error distance from initial position

Appendix D

Programming Notes: Python vs C++

This project was conceived as a demonstration of pragmatic engineering. In contrast to many PhD projects which aspire to advance the state of knowledge in regard to a particular theory or equation or technique, this project is focused on real-world performance. It must sometimes give priority to lower-order concerns that would be willed out of existence in purely theoretical research. Impurities in data such as vibration, image noise, and measurement and estimation error, cannot be removed from a physical existence simply by assuming that they don't exist.

Against this backdrop, the project must balance the concerns of producing interesting and novel work, with the necessity of overcoming obstacles to physical demonstration. A rough timeline was developed at the beginning of the project, with a three-way split between Design, Testing, and Documentation. This early deadline for design and implementation decisions - 1/3 of the way through the project - proved to be essential in reserving enough time to debug and make flight-worthy this complex and novel system.

An early decision was made to implement the Visual Navigation System in Python. As a popular, high-level programming language, this seemed to be a reasonable choice that would minimize the time spent in implementation details of low-level programming languages, allowing focus to remain on the "interesting" parts of the project - the algorithms and data analysis. Python also enjoys widespread popularity from software developers, which minimized the concern that this would lead to an

obscure and unsupportable end-product. An initial version of the system was written in Python, and was flight tested. Results were mixed: the system did function, using AprilTag detection and ZED motion tracking to compute the aircraft state. However it was catastrophically slow, averaging under two frames per second with large variances in frame times. This poor performance was ultimately tracked down to multi-threading issues, and specifically the known bottleneck in Python called the Global Interpreter Lock (GIL). The GIL is the concept that all threads of a Python program share a single instance of the Python interpreter, and must serialize their access to it. Therefore, although multiple threads can run lines of code in parallel, they must take turns decoding those lines of code. This has the effect of serializing multiple threads of execution, removing much of the benefit of multiple threads. After confirming this issue through examinations of the system load during operation (among other telemetry techniques), and after multiple attempts to redesign the system to achieve better multi-core performance, the difficult decision was made to drop the Python code base, and restart the project in C++.

Rewriting the system in C++ proved to be a valuable decision. System performance leapt further than expected, up to a full 30 frames per second under similar conditions. Additionally, this second pass at the code base afforded an opportunity to redesign and refactor the existing design work, resulting in stronger software organization.

Finally, it was observed that the "developer overhead" of working in a low-level language was not as bad as feared, either in time spent or lines of code created that would be avoided in a high-level language. Many libraries functions such as the OpenCV, AprilTag, and ZED functions used throughout the system have identical interfaces in both languages. In native code segments, C++'s more verbose code lead to additional type safety and readability. As a project grows, its readability becomes increasing important as developers have to continually refresh themselves on the code and interfaces within.

Bibliography

[1] 14:00-17:00. *ISO/IEC 16023:2000*. en. URL: https://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/02/98/29835.html (visited on 09/21/2020).

[2] Zeyad Abd Algfoor, Mohd Shahrizal Sunar, and Hoshang Kolivand. *A Comprehensive Study on Pathfinding Techniques for Robotics and Video Games*. en. Review Article. ISSN: 1687-7047 Pages: e736138 Publisher: Hindawi Volume: 2015. Apr. 2015. DOI: https://doi.org/10.1155/2015/736138. URL: https://www.hindawi.com/journals/ijcgt/2015/736138/ (visited on 09/25/2020).

[3] R. Acuna, Z. Li, and V. Willert. "MOMA: Visual Mobile Marker Odometry". In: *2018 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*. Sept. 2018, pp. 206–212. DOI: 10.1109/IPIN.2018.8533685.

[4] Ramon Andreu Altava et al. "Flight Management System Pathfinding Algorithm for Automatic Vertical Trajectory Generation". In: *2018 IEEE/AIAA 37th Digital Avionics Systems Conference (DASC)*. ISSN: 2155-7209. Sept. 2018, pp. 1–9. DOI: 10.1109/DASC.2018.8569254.

[5] *ARToolKit Home Page*. URL: http://www.hitl.washington.edu/artoolkit/ (visited on 09/21/2020).

[6] Tim Babb. *How a Kalman filter works, in pictures | Bzarg*. en-US. URL: https://www.bzarg.com/p/how-a-kalman-filter-works-in-pictures/ (visited on 09/18/2020).

[7] Jan Bacik et al. "Autonomous flying with quadrocopter using fuzzy control and ArUco markers". en. In: *Intelligent Service Robotics* 10.3 (July 2017), pp. 185–194. ISSN: 1861-2784. DOI: 10.1007/s11370-017-0219-8. URL: https://doi.org/10.1007/s11370-017-0219-8 (visited on 09/21/2020).

[8] John T. Betts. "Survey of Numerical Methods for Trajectory Optimization". In: *Journal of Guidance, Control, and Dynamics* 21.2 (1998). Publisher: American Institute of Aeronautics and Astronautics _eprint: https://doi.org/10.2514/2.4231, pp. 193–207. DOI: 10.2514/2.4231. URL: https://doi.org/10.2514/2.4231 (visited on 09/30/2020).

[9] Adi Botea. "Ultra-Fast Optimal Pathfinding without Runtime Search". en. In: (), p. 6.

[10] Adi Botea, Martin Müller, and Jonathan Schaeffer. "Near optimal hierarchical path-finding". In: *Journal of Game Development* 1 (2004), pp. 7–28.

[11] R. G. Brown. "Integrated Navigation Systems and Kalman Filtering: A Perspective". en. In: *Navigation* 19.4 (Dec. 1972), pp. 355–362. ISSN: 00281522. DOI: 10.1002/j.2161-4296.1972.tb01706.x. URL: http://doi.wiley.com/10.1002/j.2161-4296.1972.tb01706.x (visited on 09/14/2020).

[12] B. Brzozowski and K. Kaźmierczak. "Magnetic field mapping as a support for UAV indoor navigation system". In: *2017 IEEE International Workshop on Metrology for AeroSpace (MetroAeroSpace)*. June 2017, pp. 583–588. DOI: 10.1109/MetroAeroSpace.2017.7999535.

[13] B. Brzozowski et al. "A concept of UAV indoor navigation system based on magnetic field measurements". In: *2016 IEEE Metrology for Aerospace (MetroAeroSpace)*. June 2016, pp. 636–640. DOI: 10.1109/MetroAeroSpace.2016.7573291.

[14] V. Bulitko and G. Lee. "Learning in Real-Time Search: A Unifying Framework". In: *Journal of Artificial Intelligence Research* 25 (Feb. 2006). arXiv: 1110.4076, pp. 119–157. ISSN: 1076-9757. DOI: 10.1613/jair.1789. URL: http://arxiv.org/abs/1110.4076 (visited on 10/02/2020).

[15] Cesar Cadena et al. "Past, Present, and Future of Simultaneous Localization And Mapping: Towards the Robust-Perception Age". In: *IEEE Transactions on Robotics* 32.6 (Dec. 2016). arXiv: 1606.05830, pp. 1309–1332. ISSN: 1552-3098, 1941-0468. DOI: 10.1109/TRO.2016.2624754. URL: http://arxiv.org/abs/1606.05830 (visited on 08/12/2019).

[16] E. Cardenaz and J. G. Ramirez-Torres. "Autonomous navigation of unmanned aerial vehicles guided by visual features of the terrain". In: *2015 12th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE)*. Oct. 2015, pp. 1–6. DOI: 10.1109/ICEEE.2015.7357976.

[17] Joseph Carsten et al. "Global planning on the Mars Exploration Rovers: Software integration and surface testing". en. In: *Journal of Field Robotics* 26.4 (2009). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.20287, pp. 337–357. ISSN: 1556-4967. DOI: 10.1002/rob.20287. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.20287 (visited on 09/25/2020).

[18] Danny Z Chen, Robert J Szczerba, and John J Uhran Jr. "USING FRAMED-QUADTREES TO FIND CONDITIONAL SHORTEST PATHS IN AN UNKNOWN 2-D ENVIRONMENT". en. In: (), p. 33.

[19] D.Z. Chen, R.J. Szczerba, and J.J. Uhran. "Planning conditional shortest paths through an unknown environment: a framed-quadtree approach". In: *Proceedings 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human Robot Interaction and Cooperative Robots*. Vol. 3. Aug. 1995, 33–38 vol.3. DOI: 10.1109/IROS.1995.525858.

[20] Qi Cheng and Pascal Bondon. "A new unscented particle filter". In: *2008 IEEE International Conference on Acoustics, Speech and Signal Processing*. ISSN: 2379-190X. Mar. 2008, pp. 3417–3420. DOI: 10.1109/ICASSP.2008.4518385.

[21] Mark P. DeAngelo and Joseph F. Horn. "Aerial Vehicle Localization Using Generic Landmarks". In: *AIAA Guidance, Navigation, and Control Conference*. AIAA SciTech Forum. American Institute of Aeronautics and Astronautics, Jan. 2017. DOI: 10.2514/6.2017-1028. URL: https://arc.aiaa.org/doi/10.2514/6.2017-1028 (visited on 08/22/2019).

[22] Decawave. *DW1000 Datasheet*. en-US. URL: https://www.decawave.com/dw1000/datasheet/ (visited on 08/26/2019).

[23] Douglas Demyen and Michael Buro. "Efficient triangulation-based pathfinding". In: *Proceedings of the 21st national conference on Artificial intelligence - Volume 1*. AAAI'06. Boston, Massachusetts: AAAI Press, July 2006, pp. 942–947. ISBN: 978-1-57735-281-5. (Visited on 09/26/2020).

[24] V. Di Lecce, A. Amato, and V. Piuri. "Neural technologies for increasing the GPS position accuracy". In: *2008 IEEE International Conference on Computational Intelligence for Measurement Systems and Applications*. ISSN: 2159-1555. July 2008, pp. 4–8. DOI: 10.1109/CIMSA.2008.4595822.

[25] E W Dijkstra. "A Note on Two Problems in Connexion with Graphs". en. In: *Numerische Mathematik* (1959), p. 3.

[26] *DJI - The World Leader in Camera Drones/Quadcopters for Aerial Photography*. Library Catalog: www.dji.com. URL: https://www.dji.com/flame-wheel-arf (visited on 05/27/2020).

[27] *Drones and Industry 4.0*. en-us. Feb. 2019. URL: https://www.automationworld.com/factory/iiot/blog/13319582/drones-and-industry-40 (visited on 09/10/2020).

[28] Hao Du et al. "Real-Time Onboard 3D State Estimation of an Unmanned Aerial Vehicle in Multi-Environments Using Multi-Sensor Data Fusion". eng. In: *Sensors (Basel, Switzerland)* 20.3 (Feb. 2020). ISSN: 1424-8220. DOI: 10.3390/s20030919.

[29] Le Minh Duc, Amandeep Singh Sidhu, and Narendra S. Chaudhari. *Hierarchical Pathfinding and AI-Based Learning Approach in Strategy Game Design*. en. Research Article. ISSN: 1687-7047 Pages: e873913 Publisher: Hindawi Volume: 2008. June 2008. DOI: https://doi.org/10.1155/2008/873913. URL: https://www.hindawi.com/journals/ijcgt/2008/873913/ (visited on 09/30/2020).

[30] Joel Elmore. "Design of an All-In-One Embedded Flight Control System". In: *Theses and Dissertations* (Jan. 2015). URL: https://scholarscompass.vcu.edu/etd/3981.

[31] *Error Covariance Matrix - an overview | ScienceDirect Topics*. URL: https://www.sciencedirect.com/topics/engineering/error-covariance-matrix (visited on 09/18/2020).

[32] Andrew J. Fabian, Robert Klenke, and Peter Truslow. "Improving UAV-Based Target Geolocation Accuracy through Automatic Camera Parameter Discovery". In: *AIAA Scitech 2020 Forum*. American Institute of Aeronautics and Astronautics. DOI: 10.2514/6.2020-2201. URL: https://arc.aiaa.org/doi/abs/10.2514/6.2020-2201 (visited on 07/13/2020).

[33] Daquan Feng et al. "Kalman-Filter-Based Integration of IMU and UWB for High-Accuracy Indoor Positioning and Navigation". In: *IEEE Internet of Things Journal* 7.4 (Apr. 2020). Conference Name: IEEE Internet of Things Journal, pp. 3133–3146. ISSN: 2327-4662. DOI: 10.1109/JIOT.2020.2965115.

[34] Garret D Fett. "Aircraft Route Optimization using the A-Star Algorithm". en. In: (), p. 68.

[35] M. Fiala. "ARTag, a fiducial marker system using digital techniques". In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. Vol. 2. ISSN: 1063-6919. June 2005, 590–596 vol. 2. DOI: 10.1109/CVPR.2005.74.

[36] Wei Gao, Ya Zhang, and Jianguo Wang. "A strapdown interial navigation system/Beidou/Doppler velocity log integrated navigation algorithm based on a Cubature Kalman filter". eng. In: *Sensors (Basel, Switzerland)* 14.1 (Jan. 2014), pp. 1511–1527. ISSN: 1424-8220. DOI: 10.3390/s140101511.

[37] Sergio Garrido-Jurado et al. "Automatic generation and detection of highly reliable fiducial markers under occlusion". In: *Pattern Recognit.* (2014). DOI: 10.1016/j.patcog.2014.01.005.

[38] Steven Gillijns and Bart De Moor. "Unbiased minimum-variance input and state estimation for linear discrete-time systems". en. In: *Automatica* 43.1 (Jan. 2007), pp. 111–116. ISSN: 0005-1098. DOI: 10.1016/j.automatica.2006.08.002. URL: http://www.sciencedirect.com/science/article/pii/S0005109806003189 (visited on 09/11/2020).

[39] Saroj Goyal, Surendra Yadav, and Manish Mathuria. "Exploring concept of QR code and its benefits in digital education system". In: *2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. Sept. 2016, pp. 1141–1147. DOI: 10.1109/ICACCI.2016.7732198.

[40] F. Guangrui and W. Geng. "Vision-based autonomous docking and re-charging system for mobile robot in warehouse environment". In: *2017 2nd International Conference on Robotics and Automation Engineering (ICRAE)*. Dec. 2017, pp. 79–83. DOI: 10.1109/ICRAE.2017.8291357.

[41] An Guo et al. "Low-Cost Sensors State Estimation Algorithm for a Small Hand-Launched Solar-Powered UAV". In: *Sensors (Basel, Switzerland)* 19.21 (Oct. 2019). ISSN: 1424-8220. DOI: 10.3390/s19214627. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6864480/ (visited on 09/11/2020).

[42] Daniel Harabor and Adi Botea. "Hierarchical path planning for multi-size agents in heterogeneous environments". In: *2008 IEEE Symposium On Computational Intelligence and Games*. ISSN: 2325-4289. Dec. 2008, pp. 258–265. DOI: 10.1109/CIG.2008.5035648.

[43] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (July 1968). Conference Name: IEEE Transactions on Systems Science and Cybernetics, pp. 100–107. ISSN: 2168-2887. DOI: 10.1109/TSSC.1968.300136.

[44] Ralf Herrmann, Alexander Moortgat-Pick, and Steffen Marx. "Vibration Analysis of Structures using a Drone (UAV) based Mobile Sensing Platform". en. In: (), p. 8.

[45] Walter T. Higgins. "A Comparison of Complementary and Kalman Filtering". In: *IEEE Transactions on Aerospace and Electronic Systems* AES-11.3 (May 1975). Conference Name: IEEE Transactions on Aerospace and Electronic Systems, pp. 321–325. ISSN: 1557-9603. DOI: 10.1109/TAES.1975.308081.

[46] Minh-Duc Hua et al. "Implementation of a Nonlinear Attitude Estimator for Aerial Robotic Vehicles". In: *IEEE Transactions on Control Systems Technology*

22.1 (Jan. 2014). Conference Name: IEEE Transactions on Control Systems Technology, pp. 201–213. ISSN: 1558-0865. DOI: 10.1109/TCST.2013.2251635.

[47] J. Hyun et al. "UWB-based Indoor Localization Using Ray-tracing Algorithm". In: *2019 16th International Conference on Ubiquitous Robots (UR)*. June 2019, pp. 98–101. DOI: 10.1109/URAI.2019.8768568.

[48] Muhammad Ilyas et al. "Integrated Navigation System Design for Micro Planetary Rovers: Comparison of Absolute Heading Estimation Algorithms and Nonlinear Filtering". eng. In: *Sensors (Basel, Switzerland)* 16.5 (May 2016). ISSN: 1424-8220. DOI: 10.3390/s16050749.

[49] Tariqul Islam et al. "Comparison of complementary and Kalman filter based data fusion for attitude heading reference system". en. In: Dhaka, Bangladesh, 2017, p. 020002. DOI: 10.1063/1.5018520. URL: http://aip.scitation.org/doi/abs/10.1063/1.5018520 (visited on 09/11/2020).

[50] Kenneth Jensen. "Generalized Nonlinear Complementary Attitude Filter". In: *arXiv:1110.0274 [math]* (Oct. 2011). arXiv: 1110.0274. URL: http://arxiv.org/abs/1110.0274 (visited on 09/11/2020).

[51] Y. Jung, D. Lee, and H. Bang. "Study on ellipse fitting problem for vision-based autonomous landing of an UAV". In: *2014 14th International Conference on Control, Automation and Systems (ICCAS 2014)*. Oct. 2014, pp. 1631–1634. DOI: 10.1109/ICCAS.2014.6987819.

[52] Eshaan M. Khanapuri and Rajnikant Sharma. "Uncertainty Aware Geo-localization of Multi-Targets with Multi-UAV using Neural Network and Extended Kalman Filter". In: *AIAA Scitech 2019 Forum*. AIAA SciTech Forum. American Institute of Aeronautics and Astronautics, Jan. 2019. DOI: 10.2514/6.2019-1690. URL: https://arc.aiaa.org/doi/10.2514/6.2019-1690 (visited on 06/11/2019).

[53] Nak Yong Ko et al. "Features of Invariant Extended Kalman Filter Applied to Unmanned Aerial Vehicle Navigation". eng. In: *Sensors (Basel, Switzerland)* 18.9 (Aug. 2018). ISSN: 1424-8220. DOI: 10.3390/s18092855.

[54] Sven Koenig and et al. *Fast Replanning for Navigation in Unknown Terrain*. 2002.

[55] Richard E. Korf. "Depth-first iterative-deepening". en. In: *Artificial Intelligence* 27.1 (Sept. 1985), pp. 97–109. ISSN: 00043702. DOI: 10.1016/0004-3702(85)90084-0. URL: https://linkinghub.elsevier.com/retrieve/pii/0004370285900840 (visited on 09/30/2020).

[56] Richard E. Korf. "Real-time heuristic search". en. In: *Artificial Intelligence* 42.2-3 (Mar. 1990), pp. 189–211. ISSN: 00043702. DOI: 10.1016/0004-3702(90)90054-4. URL: https://linkinghub.elsevier.com/retrieve/pii/0004370290900544 (visited on 09/26/2020).

[57] Alex Kring, Alex J. Champandard, and Nick Samarin. "DHPA* and SHPA*: efficient hierarchical pathfinding in dynamic and static game worlds". In: *Proceedings of the Sixth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. AIIDE'10. Stanford, California, USA: AAAI Press, Oct. 2010, pp. 39–44. (Visited on 09/26/2020).

[58] K. Li et al. "Self-positioning for UAV indoor navigation based on 3D laser scanner, UWB and INS". In: *2016 IEEE International Conference on Information and Automation (ICIA)*. Aug. 2016, pp. 498–503. DOI: 10.1109/ICInfA.2016.7831874.

[59] Maxim Likhachev et al. "Anytime Dynamic A*: An Anytime, Replanning Algorithm". en. In: (), p. 10.

[60] Giuseppe Loianno et al. "Estimation, Control, and Planning for Aggressive Flight With a Small Quadrotor With a Single Camera and IMU". In: *IEEE Robotics and Automation Letters* 2.2 (Apr. 2017). Conference Name: IEEE Robotics and Automation Letters, pp. 404–411. ISSN: 2377-3766. DOI: 10.1109/LRA.2016.2633290.

[61] Randy MacKay. *ArduPilot + Openkai + ZED for non-GPS navigation*. en-US. Apr. 2017. URL: https://discuss.ardupilot.org/t/ardupilot-openkai-zed-for-non-gps-navigation/16308 (visited on 08/27/2019).

[62] Zouhair Mahboubi et al. "Camera Based Localization for Autonomous UAV Formation Flight". In: *Infotech@Aerospace 2011*. Infotech@Aerospace Conferences. American Institute of Aeronautics and Astronautics, Mar. 2011. DOI:

10.2514/6.2011-1658. URL: https://arc.aiaa.org/doi/10.2514/6.2011-1658 (visited on 08/22/2019).

[63] Robert Mahony, Tarek Hamel, and Jean-Michel Pflimlin. "Nonlinear Complementary Filters on the Special Orthogonal Group". In: *IEEE Transactions on Automatic Control* 53.5 (June 2008). Conference Name: IEEE Transactions on Automatic Control, pp. 1203–1218. ISSN: 1558-2523. DOI: 10.1109/TAC.2008.923738.

[64] Panos Marantos, Yannis Koveos, and Kostas J. Kyriakopoulos. "UAV State Estimation Using Adaptive Complementary Filters". In: *IEEE Transactions on Control Systems Technology* 24.4 (July 2016). Conference Name: IEEE Transactions on Control Systems Technology, pp. 1214–1226. ISSN: 1558-0865. DOI: 10.1109/TCST.2015.2480012.

[65] Héctor García de Marina, Felipe Espinosa, and Carlos Santos. "Adaptive UAV attitude estimation employing unscented Kalman Filter, FOAM and low-cost MEMS sensors". eng. In: *Sensors (Basel, Switzerland)* 12.7 (2012), pp. 9566–9585. ISSN: 1424-8220. DOI: 10.3390/s120709566.

[66] Adam Marut, Konrad Wojtowicz, and Krzysztof Falkowski. "ArUco markers pose estimation in UAV landing aid system". In: *2019 IEEE 5th International Workshop on Metrology for AeroSpace (MetroAeroSpace)*. ISSN: 2575-7490. June 2019, pp. 261–266. DOI: 10.1109/MetroAeroSpace.2019.8869572.

[67] A. Masselli et al. "A cross-platform comparison of visual marker based approaches for autonomous flight of quadrocopters". In: *2013 International Conference on Unmanned Aircraft Systems (ICUAS)*. May 2013, pp. 685–693. DOI: 10.1109/ICUAS.2013.6564749.

[68] Jefferson McBride. "Flight Control System for Small High-Performance UAVs". In: *Theses and Dissertations* (May 2010). DOI: https://doi.org/10.25772/8T74-GX16. URL: https://scholarscompass.vcu.edu/etd/2294.

[69] Pierre Merriaux et al. "A Study of Vicon System Positioning Performance". eng. In: *Sensors (Basel, Switzerland)* 17.7 (July 2017). ISSN: 1424-8220. DOI: 10.3390/s17071591.

[70] H. Merzić et al. "Map quality evaluation for visual localization". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. May 2017, pp. 3200–3206. DOI: 10.1109/ICRA.2017.7989363.

[71] M. Kara Mohamed, S. Patra, and A. Lanzon. "Designing simple indoor navigation system for UAVs". In: *2011 19th Mediterranean Conference on Control Automation (MED)*. June 2011, pp. 1223–1228. DOI: 10.1109/MED.2011.5983054.

[72] Raul Mur-Artal, J. M. M. Montiel, and Juan D. Tardos. "ORB-SLAM: a Versatile and Accurate Monocular SLAM System". In: *IEEE Transactions on Robotics* 31.5 (Oct. 2015). arXiv: 1502.00956, pp. 1147–1163. ISSN: 1552-3098, 1941-0468. DOI: 10.1109/TRO.2015.2463671. URL: http://arxiv.org/abs/1502.00956 (visited on 09/23/2020).

[73] L. Naimark and E. Foxlin. "Circular data matrix fiducial system and robust image processing for a wearable vision-inertial self-tracker". In: *Proceedings. International Symposium on Mixed and Augmented Reality*. Oct. 2002, pp. 27–36. DOI: 10.1109/ISMAR.2002.1115065.

[74] Takuma Nakamura et al. "Vision Sensor Fusion for Autonomous Landing". In: *AIAA Information Systems-AIAA Infotech @ Aerospace*. AIAA SciTech Forum. American Institute of Aeronautics and Astronautics, Jan. 2017. DOI: 10.2514/6.2017-0674. URL: https://arc.aiaa.org/doi/10.2514/6.2017-0674 (visited on 08/22/2019).

[75] *NASA - State Estimation*. en. Publisher: Brian Dunbar. URL: https://www.nasa.gov/centers/ames/research/technology-onepagers/state-estimation.html (visited on 09/21/2020).

[76] H.-Q.-T. Ngo et al. "Experimental comparison of Complementary filter and Kalman filter design for low-cost sensor in quadcopter". In: *2017 International Conference on System Science and Engineering (ICSSE)*. ISSN: 2325-0925. July 2017, pp. 488–493. DOI: 10.1109/ICSSE.2017.8030922.

[77] Thanh Binh Ngo, Hung Lan Le, and Thanh Hai Nguyen. "Survey of Kalman Filters and Their Application in Signal Processing". In: *2009 International Conference on Artificial Intelligence and Computational Intelligence*. Vol. 3. Nov. 2009, pp. 335–339. DOI: 10.1109/AICI.2009.97.

[78] Nvidia. *Jetson TX2 Module | NVIDIA Developer*. Aug. 2019. URL: https://developer.nvidia.com/embedded/jetson-tx2 (visited on 08/09/2019).

[79] Denis Oberkampf, Daniel F. DeMenthon, and Larry S. Davis. "Iterative Pose Estimation Using Coplanar Feature Points". In: *Computer Vision and Image Understanding* 63.3 (May 1996), pp. 495–511. ISSN: 1077-3142. DOI: 10.1006/cviu.1996.0037. URL: https://doi.org/10.1006/cviu.1996.0037 (visited on 08/19/2020).

[80] E. Olson. "AprilTag: A robust and flexible visual fiducial system". In: *2011 IEEE International Conference on Robotics and Automation*. May 2011, pp. 3400–3407. DOI: 10.1109/ICRA.2011.5979561.

[81] Edwin Olson. "AprilTag: A robust and flexible visual fiducial system". In: *2011 IEEE International Conference on Robotics and Automation*. ISSN: 1050-4729. May 2011, pp. 3400–3407. DOI: 10.1109/ICRA.2011.5979561.

[82] M Ostendorp. *Airborne Inventory and Inspection of Transmission Lines*. en. Tech. rep. EPRI, 2000, p. 128.

[83] C.B. Owen, Fan Xiao, and P. Middlin. "What is the best fiducial?" In: *The First IEEE International Workshop Agumented Reality Toolkit,* Sept. 2002, 8 pp.–. DOI: 10.1109/ART.2002.1107021.

[84] R. P. Padhy et al. "Monocular Vision Aided Autonomous UAV Navigation in Indoor Corridor Environments". In: *IEEE Transactions on Sustainable Computing* 4.1 (Jan. 2019), pp. 96–108. ISSN: 2377-3782. DOI: 10.1109/TSUSC.2018.2810952.

[85] Kaveh Pahlavan et al. "An Overview of Wireless Indoor Geolocation Techniques and Systems". en. In: *Mobile and Wireless Communications Networks*. Ed. by Cambyse Guy Omidyar. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2000, pp. 1–13. ISBN: 978-3-540-45494-6.

[86] Nuria Pelechano and Carlos Fuentes. "Hierarchical path-finding for Navigation Meshes (HNA∗)". en. In: *Computers & Graphics* 59 (Oct. 2016), pp. 68–78. ISSN: 0097-8493. DOI: 10.1016/j.cag.2016.05.023. URL: http://www.sciencedirect.com/science/article/pii/S0097849316300668 (visited on 09/30/2020).

[87] Katharina Pentenrieder, Peter Meier, and Gudrun Klinker. "Analysis of Tracking Accuracy for Single-Camera Square-Marker-Based Tracking". en. In: (), p. 15.

[88] F. J. Perez-Grau et al. "Multi-modal mapping and localization of unmanned aerial robots based on ultra-wideband and RGB-D sensing". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Sept. 2017, pp. 3495–3502. DOI: 10.1109/IROS.2017.8206191.

[89] R. Polvara et al. "Towards autonomous landing on a moving vessel through fiducial markers". In: *2017 European Conference on Mobile Robots (ECMR)*. Sept. 2017, pp. 1–6. DOI: 10.1109/ECMR.2017.8098671.

[90] *Pozyx*. en-US. URL: https://www.pozyx.io/?gclid=EAIaIQobChMIk-nI6oCh5AIVCa_ICh32tgiPEAAYASAAEgI9PvD_BwE (visited on 08/26/2019).

[91] Liling Ren et al. "Small Unmanned Aircraft System (sUAS) Trajectory Modeling in Support of UAS Traffic Management (UTM)". en. In: American Institute of Aeronautics and Astronautics, June 2017. ISBN: 978-1-62410-508-1. DOI: 10.2514/6.2017-4268. URL: https://arc.aiaa.org/doi/10.2514/6.2017-4268 (visited on 06/20/2018).

[92] Brent A Renfro. "An Analysis of Global Positioning System (GPS) Standard Positioning Service (SPS) Performance for 2018". en. In: (), p. 110.

[93] Eran Rippel, Aharon Bar-Gill, and Nahum Shimkin. "Fast Graph-Search Algorithms for General-Aviation Flight Trajectory Generation". In: *Journal of Guidance, Control, and Dynamics* 28.4 (2005). Publisher: American Institute of Aeronautics and Astronautics _eprint: https://doi.org/10.2514/1.7370, pp. 801–811. DOI: 10.2514/1.7370. URL: https://doi.org/10.2514/1.7370 (visited on 09/30/2020).

[94] Flyability SA. *Drone Inspections: A Guide to How Drones Are Used for Inspections*. en. URL: https://www.flyability.com/drone-inspections (visited on 09/10/2020).

[95] *Safety in design during piping engineering*. URL: https://www.hydrocarbonprocessing.com/magazine/2018/february-2018/environment-and-safety/safety-in-design-during-piping-engineering (visited on 09/10/2020).

[96] Mohammad Fattahi Sani and Ghader Karimian. "Automatic navigation and landing of an indoor AR. drone quadrotor using ArUco marker and inertial sensors". In: *2017 International Conference on Computer and Drone Applications (IConDA)*. Nov. 2017, pp. 102–107. DOI: 10.1109/ICONDA.2017.8270408.

[97] Jagan Sankaranarayanan, Houman Alborzi, and Hanan Samet. "Efficient query processing on spatial networks". In: *Proceedings of the 13th annual ACM international workshop on Geographic information systems*. GIS '05. New York, NY, USA: Association for Computing Machinery, Nov. 2005, pp. 200–209. ISBN: 978-1-59593-146-7. DOI: 10.1145/1097064.1097093. URL: https://doi.org/10.1145/1097064.1097093 (visited on 09/26/2020).

[98] Leo Willyanto Santoso, A. Setiawan, and Andre Kurniawan Prajogo. *Performance Analysis of Dijkstra, A* and Ant Algorithm for Finding Optimal Path Case Study : Surabaya City Map*. en. 2010. URL: /paper/Performance-Analysis-of-Dijkstra%2C-A*-and-Ant-for-%3A-Santoso-Setiawan/4379e478f1bdef3b00fee90a5782bef92ae5 (visited on 10/01/2020).

[99] F. H. SCHLEE, C. J. STANDISH, and N. F. TODA. "Divergence in the Kalman filter." In: *AIAA Journal* 5.6 (1967). Publisher: American Institute of Aeronautics and Astronautics _eprint: https://doi.org/10.2514/3.4146, pp. 1114–1120. ISSN: 0001-1452. DOI: 10.2514/3.4146. URL: https://doi.org/10.2514/3.4146 (visited on 09/18/2020).

[100] Z. Shi et al. "A Nano-Quadcopter Formation Flight System Based on UWB Indoor Positioning Technology". In: *2018 13th International Conference on Computer Science Education (ICCSE)*. Aug. 2018, pp. 1–4. DOI: 10.1109/ICCSE.2018.8468720.

[101] A. Smailagic and D. Kogan. "Location sensing and privacy in a context-aware computing environment". In: *IEEE Wireless Communications* 9.5 (Oct. 2002), pp. 10–17. ISSN: 1536-1284. DOI: 10.1109/MWC.2002.1043849.

[102] N. K. Soumya and M P. Selvan. "State estimation of observable and unobservable power systems". In: *2015 International Conference on Energy, Power and Environment: Towards Sustainable Growth (ICEPE)*. June 2015, pp. 1–5. DOI: 10.1109/EPETSG.2015.7510084.

[103] Maciej Stachura and Eric Frew. "WiFi Localization Experiments with an Unmanned Aircraft System". In: *AIAA Guidance, Navigation, and Control Conference*. Guidance, Navigation, and Control and Co-located Conferences. American Institute of Aeronautics and Astronautics, Aug. 2011. DOI: 10.2514/6.2011-6489. URL: https://arc.aiaa.org/doi/10.2514/6.2011-6489 (visited on 08/22/2019).

[104] Anthony Stentz. "Optimal and Efficient Path Planning for Unknown and Dynamic Environments". In: *International Journal of Robotics and Automation* 10 (1993), pp. 89–100.

[105] Stereolabs. *ZED Stereo Camera - Stereolabs*. URL: https://www.stereolabs.com/zed/ (visited on 08/09/2019).

[106] Nathan Sturtevant. *A Comparison of High-Level Approaches for Speeding Up Pathfinding*. Tech. rep. URL: https://scholar.google.com/scholar_lookup?title=A%20comparison%20of%20high-level%20approaches%20for%20speeding%20up%20pathfinding&author=N.%20R.%20Sturtevant%20&author=R.%20Geisberger (visited on 09/25/2020).

[107] Nathan Sturtevant and Michael Buro. "Partial pathfinding using map abstraction and refinement". In: *Proceedings of the 20th national conference on Artificial intelligence - Volume 3*. AAAI'05. Pittsburgh, Pennsylvania: AAAI Press, July 2005, pp. 1392–1397. ISBN: 978-1-57735-236-5. (Visited on 09/26/2020).

[108] C. Sylla. "Pairing human and machine-vision in industrial inspection tasks". en. In: *Control Engineering Practice* 1.1 (Feb. 1993), pp. 171–182. ISSN: 0967-0661. DOI: 10.1016/0967-0661(93)92117-M. URL: http://www.sciencedirect.com/science/article/pii/096706619392117M (visited on 09/10/2020).

[109] Cheickna Sylla and Colin G. Drury. "Signal detection for human error correction in quality control". en. In: *Computers in Industry* 26.2 (May 1995), pp. 147–159. ISSN: 0166-3615. DOI: 10.1016/0166-3615(94)00033-M. URL: http://www.sciencedirect.com/science/article/pii/016636159400033M (visited on 09/10/2020).

[110] L. Teixeira et al. "VI-RPE: Visual-Inertial Relative Pose Estimation for Aerial Vehicles". In: *IEEE Robotics and Automation Letters* 3.4 (Oct. 2018), pp. 2770–2777. ISSN: 2377-3766. DOI: 10.1109/LRA.2018.2837687.

[111] C. Teulière, E. Marchand, and L. Eck. "3-D Model-Based Tracking for UAV Indoor Localization". In: *IEEE Transactions on Cybernetics* 45.5 (May 2015), pp. 869–879. ISSN: 2168-2267. DOI: 10.1109/TCYB.2014.2337652.

[112] J. Tiemann, A. Ramsey, and C. Wietfeld. "Enhanced UAV Indoor Navigation through SLAM-Augmented UWB Localization". In: *2018 IEEE International Conference on Communications Workshops (ICC Workshops)*. May 2018, pp. 1–6. DOI: 10.1109/ICCW.2018.8403539.

[113] J. Tiemann, F. Schweikowski, and C. Wietfeld. "Design of an UWB indoor-positioning system for UAV navigation in GNSS-denied environments". In: *2015 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*. Oct. 2015, pp. 1–7. DOI: 10.1109/IPIN.2015.7346960.

[114] J. Tiemann and C. Wietfeld. "Scalable and precise multi-UAV indoor navigation using TDOA-based UWB localization". In: *2017 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*. Sept. 2017, pp. 1–7. DOI: 10.1109/IPIN.2017.8115937.

[115] Sebastian Trimpe and Raffaello D'Andrea. "Event-Based State Estimation With Variance-Based Triggering". In: *IEEE Transactions on Automatic Control* 59.12 (Dec. 2014). Conference Name: IEEE Transactions on Automatic Control, pp. 3266–3281. ISSN: 1558-2523. DOI: 10.1109/TAC.2014.2351951.

[116] Jeffrey Uhlmann. *First-Hand:The Unscented Transform*. URL: https://ethw.org/First-Hand:The_Unscented_Transform (visited on 09/18/2020).

[117] *Ultra-wideband*. en. Page Version ID: 911312580. Aug. 2019. URL: https://en.wikipedia.org/w/index.php?title=Ultra-wideband&oldid=911312580 (visited on 08/26/2019).

[118] Gerald J. Van Dalen, Daniel P. Magree, and Eric N. Johnson. "Absolute Localization using Image Alignment and Particle Filtering". In: *AIAA Guidance, Navigation, and Control Conference*. AIAA SciTech Forum. American Institute of Aeronautics and Astronautics, Jan. 2016. DOI: 10.2514/6.2016-0647. URL: https://arc.aiaa.org/doi/10.2514/6.2016-0647 (visited on 08/22/2019).

[119] Jonathan Vermette. "A Survey of Path-nding Algorithms Employing Automatic Hierarchical Abstraction". en. In: (), p. 20.

[120] V. Walter, M. Saska, and A. Franchi. "Fast Mutual Relative Localization of UAVs using Ultraviolet LED Markers". In: *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*. June 2018, pp. 1217–1226. DOI: 10.1109/ICUAS.2018.8453331.

[121] V. Walter et al. "Mutual Localization of UAVs based on Blinking Ultraviolet Markers and 3D Time-Position Hough Transform". In: *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*. Aug. 2018, pp. 298–303. DOI: 10.1109/COASE.2018.8560384.

[122] John Wang and Edwin Olson. "AprilTag 2: Efficient and robust fiducial detection". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. ISSN: 2153-0866. Oct. 2016, pp. 4193–4198. DOI: 10.1109/IROS.2016.7759617.

[123] Yutao Wang et al. "An Improved ArUco Marker for Monocular Vision Ranging". In: *2020 Chinese Control And Decision Conference (CCDC)*. ISSN: 1948-9447. Aug. 2020, pp. 2915–2919. DOI: 10.1109/CCDC49329.2020.9164176.

[124] Z. Wang, H. She, and W. Si. "Autonomous landing of multi-rotors UAV with monocular gimbaled camera on moving vehicle". In: *2017 13th IEEE International Conference on Control Automation (ICCA)*. July 2017, pp. 408–412. DOI: 10.1109/ICCA.2017.8003095.

[125] Zhuoya Wang et al. "An Improved Particle Filter Algorithm for Automated Aerial Refueling Based on Meanshift Clustering". In: *2018 IEEE CSAA Guidance, Navigation and Control Conference (CGNCC)*. Aug. 2018, pp. 1–5. DOI: 10.1109/GNCC42960.2018.9019016.

[126] Garrett Ward. "Design of a Small Form-Factor Flight Control System". In: *Theses and Dissertations* (Apr. 2014). DOI: https://doi.org/10.25772/Z11J-1Z28. URL: https://scholarscompass.vcu.edu/etd/3448.

[127] Norbert Wiener. *Extrapolation, Interpolation, and Smoothing of Stationary Time Series*. The MIT Press, 1964. ISBN: 978-0-262-73005-1.

[128] B. Y. Xing et al. "Slam algorithm for aruco landmark array based on synchronization optimization". en. In: *Journal of Physics: Conference Series* 1507 (Apr. 2020). Publisher: IOP Publishing, p. 052011. ISSN: 1742-6596. DOI: 10.1088/1742-6596/1507/5/052011. URL: https://doi.org/10.1088%2F1742-6596%2F1507%2F5%2F052011 (visited on 09/21/2020).

[129] Yue Yang et al. "A Nonlinear Double Model for Multisensor-Integrated Navigation Using the Federated EKF Algorithm for Small UAVs". eng. In: *Sensors (Basel, Switzerland)* 20.10 (May 2020). ISSN: 1424-8220. DOI: 10.3390/s20102974.

[130] Peter Yap et al. "Block A*: Database-Driven Search with Applications in Any-Angle Path-Planning". en. In: Jan. 2011. URL: https://openreview.net/forum?id=Sy-nBlWuZB (visited on 09/26/2020).

[131] *ZED Mini - Mixed-Reality Camera | Stereolabs*. URL: https://www.stereolabs.com/zed-mini/ (visited on 05/27/2020).

[132] W. K. Zegeye et al. "WiFi RSS fingerprinting indoor localization for mobile devices". In: *2016 IEEE 7th Annual Ubiquitous Computing, Electronics Mobile Communication Conference (UEMCON)*. Oct. 2016, pp. 1–6. DOI: 10.1109/UEMCON.2016.7777834.

[133] Peng Zhang et al. "Indoor Navigation for Quadrotor Using RGB-D Camera". en. In: *Proceedings of 2018 Chinese Intelligent Systems Conference*. Ed. by Yingmin Jia, Junping Du, and Weicun Zhang. Lecture Notes in Electrical Engineering. Springer Singapore, 2019, pp. 497–506. ISBN: 9789811322914.

[134] Xiang Zhang, S. Fronz, and N. Navab. "Visual marker detection and decoding in AR systems: a comparative study". In: *Proceedings. International Symposium on Mixed and Augmented Reality*. Oct. 2002, pp. 97–106. DOI: 10.1109/ISMAR.2002.1115078.

[135] Xiang Zhang, Y. Genc, and N. Navab. "Taking AR into large scale industrial environments: navigation and information access with mobile computers". In: *Proceedings IEEE and ACM International Symposium on Augmented Reality*. Oct. 2001, pp. 179–180. DOI: 10.1109/ISAR.2001.970531.

[136] G. Zhenglong, F. Qiang, and Q. Quan. "Pose Estimation for Multicopters Based on Monocular Vision and AprilTag". In: *2018 37th Chinese Control Conference (CCC)*. July 2018, pp. 4717–4722. DOI: 10.23919/ChiCC.2018.8483685.

[137] M. Zhou et al. "Indoor UAV Localization using Manifold Alignment with Mobile AP Detection". In: *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*. May 2019, pp. 1–6. DOI: 10.1109/ICC.2019.8761348.